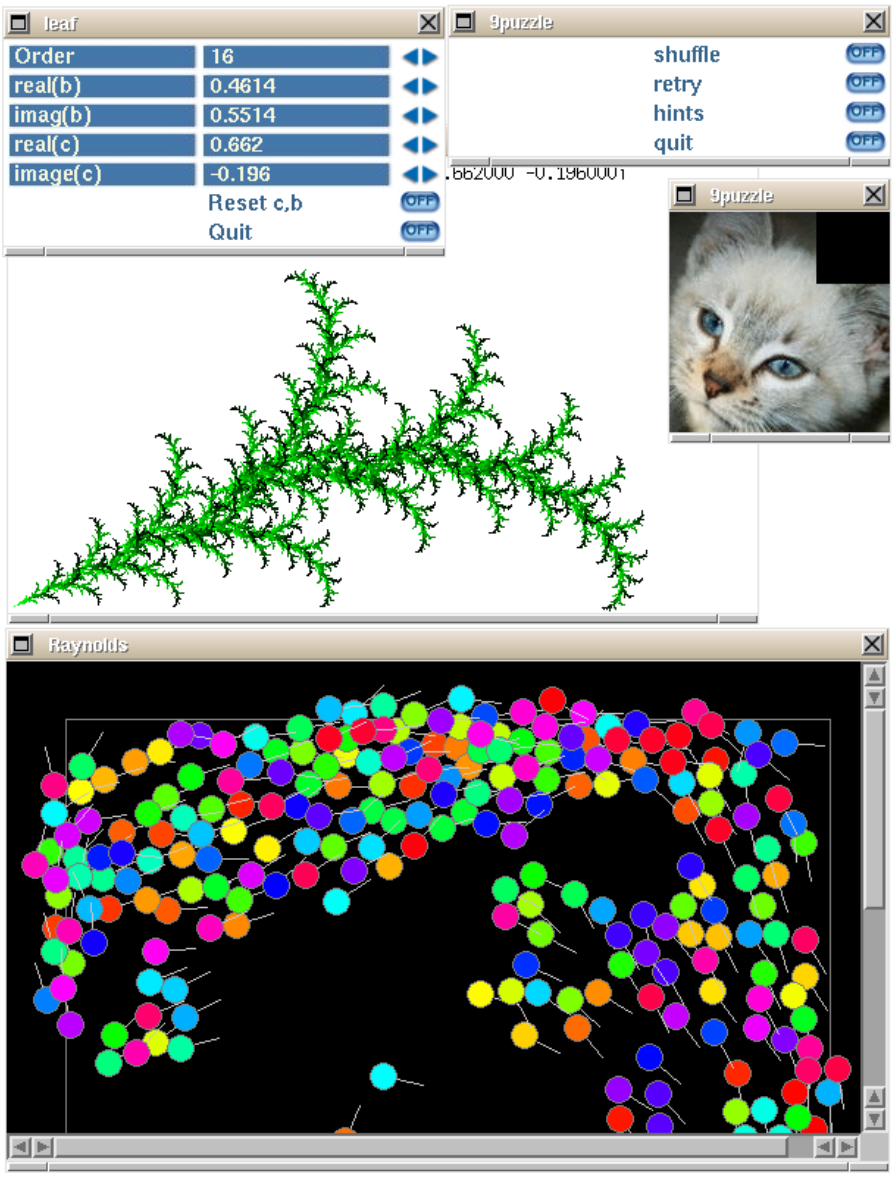
Easy and Gratifying Graphics library for X11

EGGX / ProCALL version 0.95

日本語版ユーザーズガイド

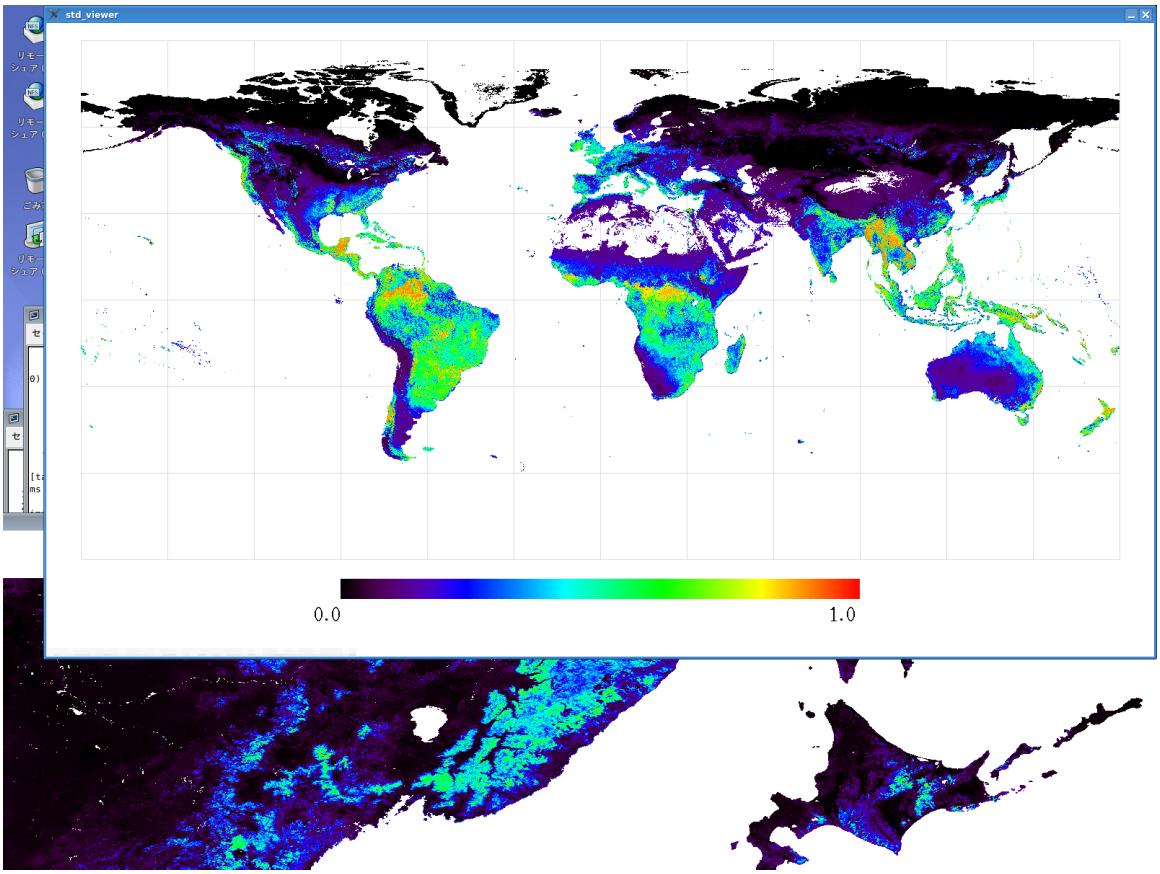
Chisato Yamauchi Aug. 28, 2021



表紙のスクリーンショットは，東京電機大学の松田様と京都産業大学の安田様のコードによるものです

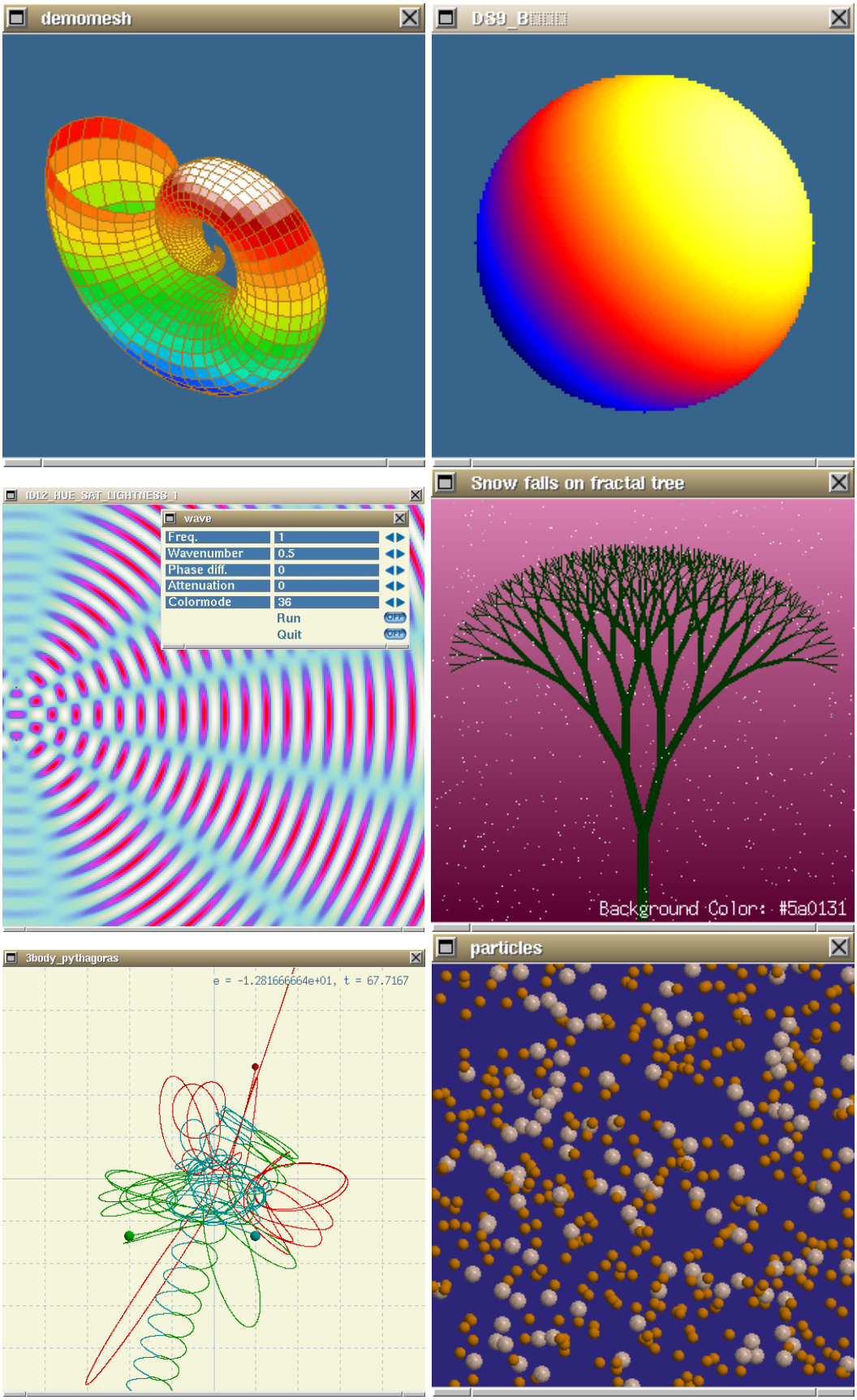
ギャラリー

EGGX を使ったソフトウェアのスクリーンショットの紹介.



リモートセンシングデータのビュアー (名古屋大学の佐々井様)

サンプルプログラム「loupe.c」



EGGX 付属のサンプルプログラム (東京電機大学の松田様)

1 はじめに

1.1 EGGX / ProCALL とは

EGGX / ProCALL (えっぐえっくす/ぷろこーる) は「**究極の簡単さ**」を目指して作り上げた簡素なグラフィックスライブラリです．C 言語か FORTRAN から各種関数 (サブルーチン) を利用する事ができ，ファミコンを贅沢にしたようなグラフィックス機能を，N88-BASIC のような感覚で操作する事ができます．C の関数群を “EGGX”(§2.2)，FORTRAN のルーチン群を “ProCALL”(§3.1) と呼んでいます．とにかく「簡単」に使えて,「8 ビットマシンの BASIC のような楽しさ」をこれからプログラミングを始める方々に伝えることができれば，と考えて作成したもので，特に初等プログラミング教育には最適なライブラリの 1 つとなるでしょう．

EGGX / ProCALL は単純なライブラリですから，専用のランタイムファイルの導入は一切不要です．もちろん，ユーザが作成した実行ファイルは，立派なアプリケーション・ソフトウェアになります．また，X11 関連のライブラリは Xlib しか使っていませんから，インストールも容易です．

1.2 EGGX / ProCALL の特徴

ユーザ関数の簡単さは常識破りです．ウィンドゥサイズだけを引数とする「たった 1 つの関数」を呼ぶだけで，X Window 上でウィンドゥを開いて，即グラフィックス関数が扱えます．ウィンドゥが隠れた場合の再描画はライブラリ側がやってくれますから，ユーザは扱いが面倒なイベントを気にする必要がないので，描画関数を並べるだけで簡単に絵を描くことができます．例えば，数値計算のコードに若干の描画関数を挟み込んで，デバッグ時などに必要な時だけ計算させながらその様子をモニターする，といった事も容易です．

任意のサイズで複数のグラフィックス用ウィンドゥをオープンし，24 ビットでのカラー指定が可能な線, 点, 多角形, 円などの描画，フォントセットの描画 (半角・全角混在の文字列もそのまま描画可能)，数値計算の可視化に必要な種々の矢印の描画，カラーバーの生成 (約 50 種類)，背景透過処理が可能な 24 ビット画像の転送機能・画面スクロール機能 (ファミコン等のスプライトとハードウェアスクロールに相当)，レイヤ機能 (PC-9801 の V-RAM の 2 プレーンに相当) によるスムーズなアニメーション，ggetch() 関数 (§2.4.48) を使ったキー入力読み込み (N88-BASIC の INKEY$に相当)，ggetevent() 関数 (§2.4.49) を使ったマウス・キー入力読み込みが可能です．

もうおわかりかと思いますが，24 ビットカラーが使えて，アニメーションができるとなれば，現代の 2D グラフィックスに必要な基本要素は十分に整っているというわけです．あとは，ユーザのプログラミング次第で，いくらでも派手な数値計算の可視化やゲーム等を作る事が可能です．

また，netpbm[1] の各種コマンド，ImageMagick[2] の convert コマンドを通して様々なフォーマットの画像を読み書きするめの関数が用意されており，これらを活用すれば画像閲覧ツールや動画作成ツールなどが簡単に作れます．

さらに，Version 0.91 以降では，スクロールバー・インタフェースを提供するようになり，ディスプレイに収まりきらないような大きな画素サイズの画像を容易に扱えるようになりました．

1.3 EGGX / ProCALL に関する情報

EGGX / ProCALL の Web ページ:

`http://www.ir.isas.jaxa.jp/~cyamauch/eggx_procall/`

では，サンプルプログラムやリリース情報の他，派生ライブラリや情報教育等での活用についてまとめています．目的に応じてご利用ください．

---

[1] http://sourceforge.net/projects/netpbm/

[2] http://www.imagemagick.org/

2　C編

2.1　チュートリアル

2.1.1　使い方の基本

ユーザプログラムの冒頭で,

```
#include <eggx.h>
```

と宣言します[3]．C 標準関数と同じように EGGX の関数を使ってプログラムを書き，コンパイルは egg コマンドを用います.

例　`egg program.c`

2.1.2　プログラム例

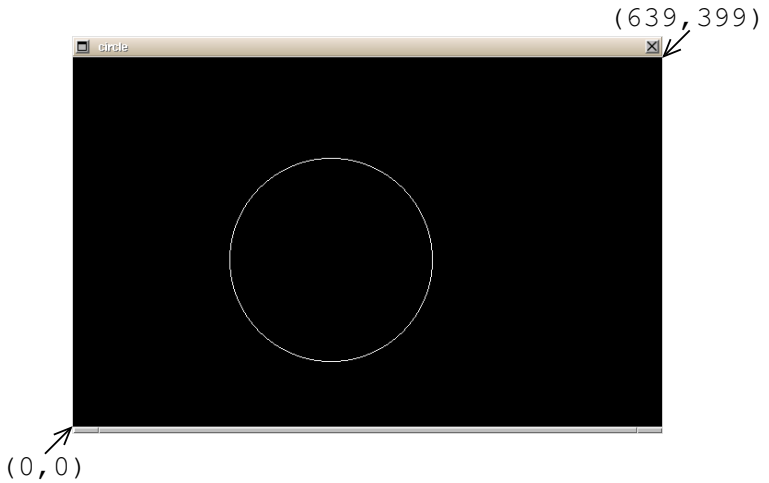
EGGX を使って円を描くプログラムを示します.

```
#include <eggx.h>                       /* EGGX を使う時に必要 */

int main()
{
    int win;                            /* ウィンドゥ番号を格納する変数 */
    win = gopen(640,400);               /* 640x400 ピクセルのグラフィックス用ウィンドゥを開く */
    circle(win, 280, 180, 110, 110);    /* 中心 (280,180)，半径 110 の円を描く */
    ggetch();                           /* キー入力があるまで待つ */
    gclose(win);                        /* グラフィックス用ウィンドゥを閉じる */
    return 0;                           /* 終了 */
}
```

ウィンドゥを開き，ウィンドゥ中央より少し左下に円を描きます．ウインドゥを閉じる前に ggetch() を使っていますが，これはプログラムが一瞬で終わってしまわないようにするためのものです.

次の絵は，実行結果のスクリーンショットです.



デフォルトではこのように，ウィンドゥの左下が座標系の原点となっています．例では円を描きましたが，点，線，多角形，シンボル，文字等も同じようにウィンドゥ番号と座標を与えて描画します.

[3] 本格的なソフトウェア開発で EGGX を利用する場合は，eggxlib.h を利用する事をお勧めします．§2.3.4をご覧ください.

2.2 EGGX の関数一覧

2.2.1 標準関数

章	関数名	機能
§2.4.1	gopen	任意のサイズのグラフィックス用ウィンドゥを開く
§2.4.2	gclose	任意のグラフィックス用ウィンドゥを閉じる
§2.4.3	gcloseall	すべてのグラフィックス用ウィンドゥを閉じ，X サーバと接続を断つ
§2.4.4	gresize	グラフィックス描画領域のサイズ変更を行なう
§2.4.5	winname	ウィンドゥのタイトルを変更する
§2.4.6	coordinate	アプリケーション座標系の変更 (参照点の座標とスケールを与える)
§2.4.7	window	アプリケーション座標系の変更 (左下と右上の座標を与える)
§2.4.8	layer	レイヤの設定を行う
§2.4.9	copylayer	レイヤのコピーを行う
§2.4.10	gsetbgcolor	ウィンドゥのバックグラウンドカラー (gclr での色) を指定する
§2.4.11	gclr	描画レイヤの全消去
§2.4.12	tclr	端末画面の消去
§2.4.13	newpen	描画色 (16 色) の変更
§2.4.14	newcolor	描画色の変更 (X サーバの持つ色を直接指定)
§2.4.15	newrgbcolor	描画色の変更 (Red,Green,Blue の輝度を指定)
§2.4.16	newhsvcolor	描画色の変更 (Hue,Saturation,Value を指定)
§2.4.17	makecolor	変数からカラーを生成する (カラーバー生成)
§2.4.18	newlinewidth	線幅の変更
§2.4.19	newlinestyle	線のスタイルの変更
§2.4.20	newgcfunction	GC ファンクションの変更
§2.4.21	pset	点の描画
§2.4.22	drawline	直線の描画
§2.4.23	moveto, lineto	連続的に直線を描く
§2.4.25	drawpts	複数の点を描く
§2.4.26	drawlines	折れ線を描く
§2.4.27	drawpoly	多角形を描く
§2.4.28	fillpoly	多角形を塗り潰す
§2.4.29	drawrect	長方形を描く
§2.4.30	fillrect	長方形の領域を塗り潰す
§2.4.31	drawcirc, circle	中心座標，半径を与えて円を描く
§2.4.32	fillcirc	中心座標，半径を与えて円を塗り潰す
§2.4.33	drawarc	円の中心，半径，始点，終点の角度を与えて円弧を描く
§2.4.34	fillarc	円の中心，半径，始点，終点の角度を与えて円弧を塗り潰す
§2.4.35	drawsym	1 個のシンボルの描画
§2.4.36	drawsyms	複数のシンボルを描く
§2.4.37	drawarrow	種々の矢印の描画
§2.4.38	newfontset	フォントセット (日本語フォント) の指定
§2.4.39	drawstr	文字列の描画
§2.4.40	gscroll	ピクセル単位で描画レイヤをスクロールする
§2.4.41	gputarea	任意のウィンドゥ・レイヤ・領域の画像を描画レイヤにコピーする
§2.4.42	gputimage	メモリバッファの (マスク付き) 画像データを描画レイヤに一括転送する
§2.4.43	ggetimage	任意のウィンドゥ・レイヤ・領域の画像データをメモリバッファに取り込む
§2.4.44	gsaveimage	任意のウィンドゥ・レイヤ・領域の画像をコンバータを通してファイルに保存する
§2.4.45	readimage	ファイルの画像データをコンバータ (netpbm 等) を通してメモリバッファに取り込む
§2.4.46	writeimage	メモリバッファの画像データをコンバータ (netpbm 等) を通してファイルに保存する
§2.4.47	gsetnonblock	`ggetch()`，`ggetevent()`，`ggetxpress()` の動作モードを設定する
§2.4.48	ggetch	キーボードから入力された文字を返す
§2.4.49	ggetevent	マウスやキーボードからの入力の情報を返す
§2.4.50	ggetxpress	マウスからのボタンクリック，キーボードからの入力の情報を返す
§2.4.51	msleep	ミリ秒単位で実行を延期する

2.2.2 Advanced 関数 (中級〜上級者向け)

章	関数名	機能
§2.5.1	ggetdisplayinfo	X サーバの情報 (depth, 画面サイズ) を取得する
§2.5.2	gsetnonflush, ggetnonflush	描画関数等におけるフラッシュに関する設定
§2.5.3	gflush	描画命令等をフラッシュする
§2.5.4	gsetinitialattributes	gopen() でのウィンドゥ属性を変更する
§2.5.5	ggetinitialattributes	gopen() でのウィンドゥ属性を取得する
§2.5.6	gsetinitialbgcolor	gopen() での背景カラーを指定する
§2.5.7	gsetborder	ウィンドゥのホーダーと色を指定する
§2.5.8	gsetinitialborder	gopen() でののウィンドゥのボーダーを指定する
§2.5.9	gsetinitialgeometry	x,y の値を含む文字列から gopen() でのウィンドゥ出現座標等を設定する
§2.5.10	gsetinitialwinname	gopen() でのウィンドゥ名, アイコン名, リソース名, クラス名を指定する
§2.5.11	gsetscrollbarkeymask	EGGX によるスクロールバーのキー操作のためのキーマスクを設定する
§2.5.12	generatecolor	変数からカラーを生成する (コントラスト，ブライトネス，γ 補正等が可能)

2.3 TIPS

2.3.1 描画速度を改善する方法

- レイヤ (ダブルバッファリング) を使う

  layer() 関数 (§2.4.8) と copylayer() 関数 (§2.4.9) とを使って，描画関数では常に非表示レイヤに描きましょう．描き終わったら，非表示レイヤを表示レイヤにコピーします (copylayer() 関数)．このようにする事で，かなり描画速度が改善されます．

  次の例は，アニメーションを描く場合の典型的なコードです．

```
#include <eggx.h>

int main()
{
    int win;
    win = gopen(640,400);
    layer(win,0,1);                 /* 表示対象をレイヤ 0，描画対象をレイヤ 1 に設定 */
    while ( 1 ) {
        gclr(win);                  /* レイヤ 1 を初期化 */
            :                       /* ←この部分に描画関数を書く */
        copylayer(win,1,0);         /* レイヤ 1 をレイヤ 0 に「瞬時に」コピー */
        msleep(10);                 /* 10 ミリ秒遅らせる（アニメーションの速度調整）*/
    }
```

- 色設定は，newcolor() 関数以外を使う．

  できるだけ newpen() 関数 (§2.4.13)，newrgbcolor() 関数 (§2.4.15) など，数値で色を指定する関数を使いましょう．

- 色設定の頻度を最小化する

  newpen() → pset() → newpen() → pset() → newpen() → pset() → newpen() → pset() →…のように頻繁に色設定を行なうと，描画パフォーマンスが得られない事があります．

  newpen() → pset() → pset() → pset() → newpen() → pset() → pset() → pset() →…のように，できるだけ同じ色ごとに描画命令をまとめるようにします．

- 非自動フラッシュを使う (上級者向き)

  §2.5.2と §2.5.3をご覧ください.

2.3.2 ウィンドゥ座標の原点 (0,0) を左上に変更するには

次のように，gsetinitialattributes() 関数 (§2.5.4) で，BOTTOM_LEFT_ORIGIN 属性を無効化してから，ウィンドゥを開きます.

例
```
gsetinitialattributes(DISABLE, BOTTOM_LEFT_ORIGIN);
```

2.3.3 C++からの利用

C++の場合，EGGX で提供しているヘッダファイルで,「extern "C" {…}」が宣言されますので，C++の場合もそのまま eggx.h をインクルードできます．コンパイル方法は C の場合と同様です．ソースファイル名のサフィックスは「.cc」としてください.

例
```
egg program.cc
```

デフォルトではコンパイラとして,「g++」が設定されています．他のコンパイラを使いたい場合は，スクリプト egg を編集してください.

2.3.4 本格的なソフトウェア開発で EGGX を利用する場合

EGGX の実際の関数名 (シンボル名) は，必ず先頭に「eggx_」がつく名称となっていますが，情報教育等での利用のしやすさを考慮して，ヘッダファイル eggx.h では，例えば

```
#define gopen eggx_gopen
```

のように，関数名を短く書けるようにするためのマクロが定義されています．しかし，こういったマクロ定義は，構造体や C++のクラスを使う場合にはトラブルを引き起こす事があります.

したがって，EGGX を本格的なソフトウェア開発で利用する場合は，次のように eggx.h のかわりに eggxlib.h を使う事をお勧めします:

```
#include <eggxlib.h>
```

eggxlib.h では関数名に対するマクロは定義されないので，実際の関数名をソースコードに記述する必要がありますが，上記のマクロ定義によって「構造体等のメンバ名」が置換されてエラーとなる，といったトラブルを避ける事ができます．この場合には，次のように，このマニュアルに書かれている関数名の先頭に「eggx_」をつけて関数を利用します.

例
```
win = eggx_goepn(800,600) ;
```

2.4 EGGX の標準関数リファレンス

2.4.1 int gopen(int xsize, int ysize)

機　能　任意のサイズのグラフィックス画面を開く

グラフィックス用ウィンドゥを開き，返り値として，EGGX で使用するウィンドゥ番号を返します．EGGX では，この番号を描画関数に与えてグラフィックスを扱います．

引数 xsize，ysize にはそれぞれ横方向，縦方向の描画領域のピクセル数を指定します．指定できる最大値は，32767 です．

デフォルトでは，ルートウィンドゥ(つまり背景) のピクセル数とほぼ同じ，あるいはそれを越えるサイズの描画領域が指定された場合には，描画領域よりも小さいウィンドゥを開きます[4]．この場合，スクロールバー・インタフェースが提供され，マウスまたはキーボードから任意の描画領域を表示する事が可能です．

使用例　`win = gopen(800,600) ;`

2.4.2 void gclose(int wn)

機　能　グラフィックス用ウィンドゥを閉じる

wn で指定されたウィンドゥを閉じます．

使用例　`gclose(win) ;`

2.4.3 void gcloseall(void)

機　能　すべてのグラフィックス用ウィンドゥを閉じ，X サーバと接続を断つ

全てのウィンドゥを閉じ，X サーバと接続を断ち，ライブラリの内部処理で利用しているメモリ領域を開放します．

使用例　`gcloseall() ;`

2.4.4 void gresize(int wn, int xsize, int ysize)

機　能　グラフィックス描画領域のサイズ変更を行なう

wn で指定されたウィンドゥの描画領域のサイズを変更します．引数 xsize，ysize にはそれぞれ横方向，縦方向のピクセル数を指定します．

デフォルトでは，描画領域と同時にウィンドゥの大きさも変更されますが，スクロールバー・インタフェースが有効な場合に gsetinitialgeometry() 関数 (§2.5.9) による指定があった場合，ウィンドゥのサイズはその指定に従います．

描画領域のサイズ変更は，デフォルトでは左下を原点として行ないますが，gsetinitialattributes() 関数 (§2.5.4) で，BOTTOM_LEFT_ORIGIN 属性が無効化されている場合は，左上が原点となります．

使用例　`gresize(win, 1280,960) ;`

[4] EGGX/ProCALL のビルド時に Xinerama が有効にされていた場合は，最小サイズのディスプレイのピクセルサイズと比較されます．

2.4.5 int winname(int wn, const char *argsformat, ...)

機　能　ウィンドゥのタイトルを変更する

gopen() で開いたウィンドゥはデフォルトではユーザの実行ファイル名がウィンドゥのタイトルとして設定されますが，このタイトルは自由に変更できます．最初の引数 wn はウィンドゥ番号で，argsformat(とそれに続く引数) で指定される文字列がタイトルとして設定されます．2 つめの引数 argsformat 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっています．使用例のように，変数の値を表示するような使い方もあります．

返り値は設定したウィンドゥタイトルの文字列の長さです．

使用例　`winname(win,"penguin x=%f y=%f",x,y) ;`

2.4.6 void coordinate(int wn, int xw, int yw, double xa, double ya, double xscale, double yscale)

機　能　アプリケーション座標系の変更 (参照点の座標とスケールを与える)

wn で指定したウィンドゥのアプリケーション座標系を変更します

ウィンドゥ上のウィンドゥ座標系 (座標値は整数) は，左下が (0, 0) で，右上が (xsize-1, ysize-1) であり，デフォルトではアプリケーション座標系 (座標値は実数) の座標値はウィンドゥ座標系のそれに一致しています．

coordinate() 関数を使う事により，アプリケーション座標系の (xa, ya) をウィンドゥ座標系の (xw, yw) に対応させ，それぞれのスケーリングファクターを xscale，yscale で指定します．すなわち，描画関数等におけるアプリケーション座標 (x, y) からウィンドゥ座標 (x, y) への変換は，次の式により行なわれる事を意味します:

$$\mathtt{x} = \mathtt{xw} + (x - \mathtt{xa}) \cdot \mathtt{xscale}$$
$$\mathtt{y} = \mathtt{yw} + (y - \mathtt{ya}) \cdot \mathtt{yscale}$$

EGGX の描画関数は，アプリケーション座標系 (座標値は実数) で指定するので，この関数を一度使えば，座標系の変換は各描画関数が自動的に行なうようになります．

次の使用例では，ウィンドゥの左下をアプリケーション座標系の (-40.0, -20.0) に対応させ，スケーリングファクターを縦・横とも 2.0 に設定します．

使用例　`coordinate(win, 0,0, -40.0,-20.0, 2.0,2.0) ;`

なお，gsetinitialattributes() 関数 (§2.5.4) で，BOTTOM_LEFT_ORIGIN 属性が無効化されている場合は，ウィンドゥ座標系の原点 (0, 0) は左上になります．

座標系の変更には，window() 関数 (§2.4.7) を使う方法もあります．ご検討ください．

2.4.7 void window(int wn, double xs, double ys, double xe, double ye)

機　能　アプリケーション座標系の変更 (左下と右上の座標を与える)

wn で指定したウィンドゥのアプリケーション座標系を変更します (実際のグラフィックスエリアの大きさが変わるわけではありません).

ウィンドゥ上のウィンドゥ座標系 (座標値は整数) は，左下が (0, 0) で，右上が (xsize-1, ysize-1) であり，デフォルトではアプリケーション座標系 (座標値は実数) の座標値はウィンドゥ座標系のそれに一致しています．

window() 関数を使う事により，アプリケーション座標系の左下 (つまりウィンドゥ座標系での (0, 0)) を (xs, ys)，右上を (xe, ye) に変更できます．

EGGX の描画関数は，アプリケーション座標系 (座標値は実数) で指定するので，この関数を一度使えば，座標系の変換は各描画関数が自動的に行なうようになります．

次の使用例では，アプリケーション座標系の左下を (-20.0, -10.0)，右上を (799.0, 599.0) に変更します．

使用例 `window(win, -20.0, -10.0, 799.0, 599.0) ;`

なお，gsetinitialattributes() 関数 (§2.5.4) で，`BOTTOM_LEFT_ORIGIN` 属性が無効化されている場合は，ウィンドゥ座標系の原点 (0, 0) は左上になります．

座標系の変更には，coordinate() 関数 (§2.4.6) を使う方法もあります．ご検討ください．

2.4.8 void layer(int wn, int lys, int lyw)

機 能 レイヤの設定をする

EGGX ではグラフィックス用ウィンドゥ毎に 8 枚のレイヤを持ち，表示するレイヤと書き込むレイヤを独立に指定できます．`wn` にはウィンドウ番号を指定し，`lys` には表示するレイヤ番号，`lyw` には書き込むレイヤ番号を 0〜7 で指定します．

現在表示しているレイヤに対して (`lys == lyw` の場合に) 連続して描画関数を実行すると，描画パフォーマンスが得られない事があります．高速な描画が必要な場合には，現在表示していないレイヤに対して描画し，copylayer() 関数 (§2.4.9) で描画レイヤの画像を表示レイヤにコピーするようにします．

デフォルトでは `layer(wn,0,0)` の状態となっています．

使用例 `layer(win,0,1) ;`

2.4.9 void copylayer(int wn, int lysrc, int lydest)

機 能 レイヤのコピーをする

`wn` のウィンドウ番号の，レイヤ `lysrc` の画像をレイヤ `lydest` にそのままコピーします．このコピーは瞬時に行われるため，アニメーションの再生に使うことができます．

使用例 `copylayer(win,1,0) ;`

2.4.10 void gsetbgcolor(int wn, const char *argsformat, ...)

機 能 ウィンドゥの背景色を変更する

`wn` で指定されたウィンドゥの背景色 (gclr() 関数 (§2.4.11) で初期化される色) を変更します．`argsformat`(とそれに続く引数) で指定される文字列を背景色に設定します．2 つめの引数 `argsformat` 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっています．この背景色の文字列には，X サーバの `rgb.txt`[5] に設定されている色か，`"#c0c0ff"`のように，16 進数の Red,Green,Blue を指定します．

使用例 `gsetbgcolor(win,"white") ;`

2.4.11 void gclr(int wn)

機 能 描画レイヤの全消去

[5] `rgb.txt` は UNIX 系の OS なら `/usr/X11R6/lib/X11/` などにあります．

wn で指定したウィンドゥの描画レイヤを gsetinitialbgcolor() 関数 (§2.5.6) あるいは gsetbgcolor() 関数 (§2.4.10) で指定した色で初期化します．gsetbgcolor() 関数，gsetinitialbgcolor() 関数での指定がない場合の色は黒となっています．

使用例　`gclr(win) ;`

2.4.12　void tclr(void)

機　能　端末のクリア
端末をクリアし，カーソルの位置をホームポジションに戻します．

使用例　`tclr() ;`

2.4.13　void newpen(int wn, int cn)

機　能　描画色の変更
wn で指定したウィンドゥでの描画色を変更します．cn と色との関係は以下の通りです．
0:黒　1:白　2:赤　3:緑　4:青　5:シアン　6:マゼンタ　7:黄
8:DimGray　9:Gray　10:red4　11:green4　12:blue4　13:cyan4　14:magenta4　15:yellow4
red4，green4…の “4” のつく色は，暗い赤，暗い緑…となっています．
デフォルトでは，白が指定されています．

使用例　`newpen(win,2) ;`

2.4.14　void newcolor(int wn, const char *argsformat, ...)

機　能　描画色の変更
wn で指定したウィンドゥでの描画色を変更します．argsformat(とそれに続く引数) で指定される文字列を描画色に設定します．2 つめの引数 argsformat 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっています．この描画色の文字列には，X サーバの rgb.txt[6] に設定されている色か，"#c0c0ff" のように，16 進数の Red,Green,Blue を指定します．

使用例　`newcolor(win,"Violet") ;`

2.4.15　void newrgbcolor(int wn, int r, int g, int b)

機　能　描画色の変更
wn で指定したウィンドゥでの描画色を変更します．r,g,b にはそれぞれ Red,Green,Blue の輝度を 256 段階の整数 (0〜255) で指定します．

使用例　`newrgbcolor(win,255,127,0) ;`

2.4.16　void newhsvcolor(int wn, int h, int s, int v)

機　能　描画色の変更

[6] rgb.txt は UNIX 系の OS なら /usr/X11R6/lib/X11/ などにあります．

wn で指定したウィンドゥでの描画色を変更します．h,s,v にはそれぞれ，Hue, Satulation, Value を指定します[7]．s と v は 256 段階の整数 (0〜255) を，h は 0〜359 までの整数 (角度) を指定します．

使用例　newhsvcolor(win,120,250,240) ;

2.4.17　int makecolor(int cmode, double dmin, double dmax, double data, int *r, int *g, int *b)

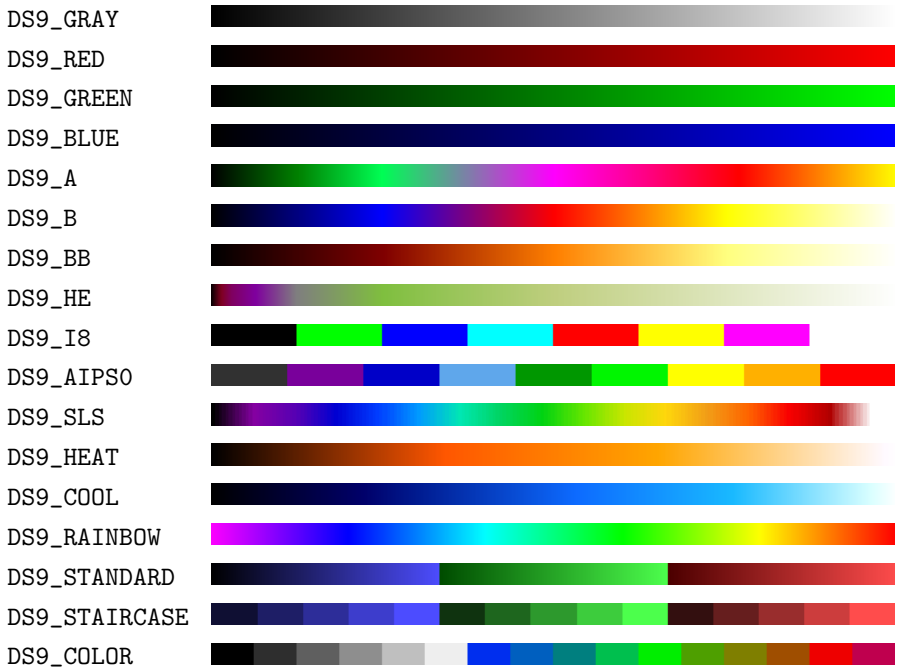
機　能　変数からカラーを生成する (カラーバー生成)

変数 data の値を使って，r,g,b それぞれに 256 段階の Red,Greed,Blue のカラーを生成します．変数の最小，最大は dmin,dmax で指定します．この関数で得た，r,g,b の値は newrgbcolor() 関数 (§2.4.15) にそのまま与えて使う事ができます．

返り値は data が dmin,dmax の範囲内にあった場合は 0 を返し，dmin 未満であれば負の値を，dmax を越えていれば，正の値を返します．

cmode はカラーパターンの番号で，以下に示す約 50 種類のカラーパターンが利用できます．DS9_GRAY 等はマクロで，実際の値は 0 から始まる整数です．詳細は，eggx_color.h を見てください．

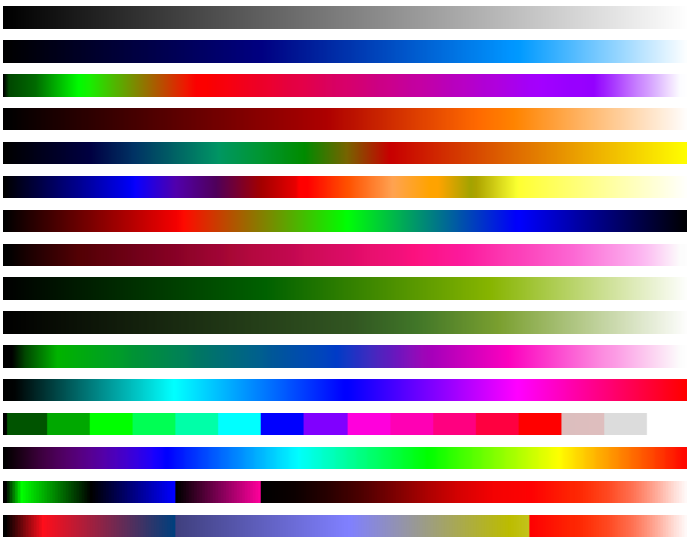
以下は fits ビューア DS9 コンパチのカラーパターンです．



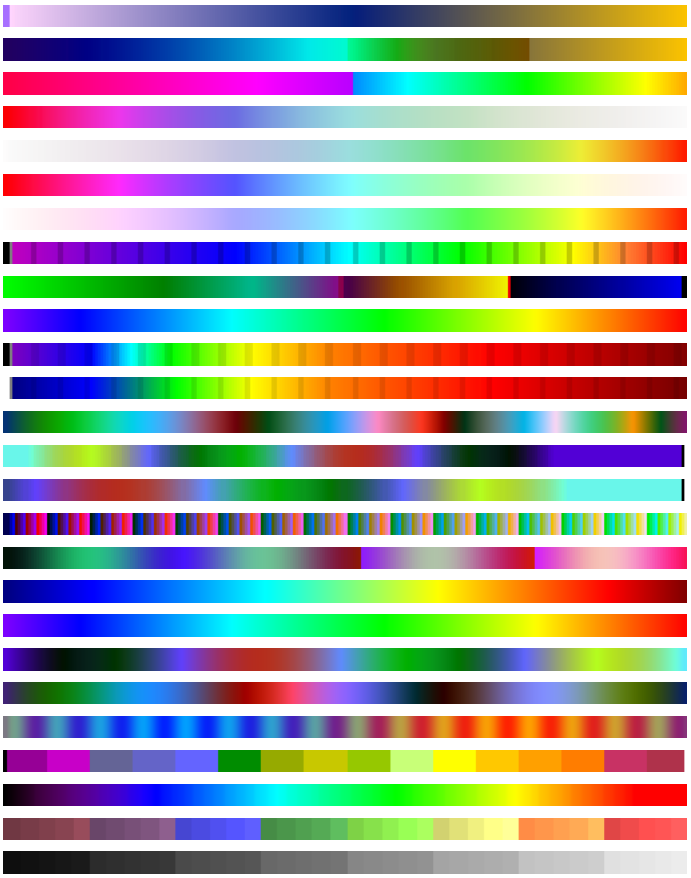
以下は可視化ソフトウェア IDL コンパチのカラーパターンです．

[7] newhsvcolor() 関数は，京都産業大学の安田様からいただきました．

IDL1_B_W_LINEAR
IDL1_BLUE_WHITE
IDL1_GRN_RED_BLU_WHT
IDL1_RED_TEMPERATURE
IDL1_BLUE_GREEN_RED_YELLOW
IDL1_STD_GAMMA_II
IDL1_PRISM
IDL1_RED_PURPLE
IDL1_GREEN_WHITE_LINEAR
IDL1_RGN_WHT_EXPONENTIAL
IDL1_GREEN_PINK
IDL1_BLUE_RED
IDL1_16_LEVEL
IDL1_RAINBOW
IDL1_STEPS
IDL1_STERN_SPECIAL



IDL2_HAZE
IDL2_BLUE_PASTEL_RED
IDL2_PASTELS
IDL2_HUE_SAT_LIGHTNESS_1
IDL2_HUE_SAT_LIGHTNESS_2
IDL2_HUE_SAT_VALUE_1
IDL2_HUE_SAT_VALUE_2
IDL2_PURPLE_RED_WITH_STRIPES
IDL2_BEACH
IDL2_MAC_STYLE
IDL2_EOS_A
IDL2_EOS_B
IDL2_HARDCANDY
IDL2_NATURE
IDL2_OCEAN
IDL2_PEPPERMINT
IDL2_PLASMA
IDL2_BLUE_RED
IDL2_RAINBOW
IDL2_BLUE_WAVES
IDL2_VALCANO
IDL2_WAVES
IDL2_RAINBOW18
IDL2__RAINBOW
IDL2_ORBIT_VIEWER_COLOR
IDL2_ORBIT_VIEWER_GRAY



使用例　`makecolor(DS9_SLS,v_min,v_max,v,&r,&g,&b) ;`

2.4.18　void newlinewidth(int wn, int width)

機　能　線幅の変更

wn で指定したウィンドゥで線を描く時の線幅を変更します．デフォルトでは幅 1 が設定されています．

この関数で線幅を変更すると，drawsym() 関数 (§2.4.35) や drawarrow() 関数 (§2.4.37) 等で描かれる図形にも影響を与えるので注意してください．

使用例　`newlinewidth(win, 2) ;`

2.4.19　void newlinestyle(int wn, int style)

機　能　線のスタイルの変更

`wn` で指定したウィンドゥで線を描く時のスタイルを変更します．引数 `style` に与える事ができる値は，`LineSolid`(実線) と `LineOnOffDash`(点線) です．デフォルトでは実線 (`LineSolid`) が設定されています．

この関数で線のスタイルを変更すると，drawsym() 関数 (§2.4.35) や drawarrow() 関数 (§2.4.37) 等で描かれる図形にも影響を与えるので注意してください．

使用例　`newlinestyle(win, LineOnOffDash) ;`

2.4.20　void newgcfunction(int wn, int fnc)

機　能　GC ファンクションの変更

`wn` で指定したウィンドゥでの描画関数 (画像の転送等も含む) を実行する時の，GC ファンクションを変更します．

GC ファンクションとは，源泉 RGB 値 (例えば newcolor() 関数で指定したカラーの RGB 値) と描画先ピクセルの RGB 値との論理演算により，描画される RGB 値を決定する仕組みです．

指定可能な GC ファンクションは次の 16 種類で，デフォルトでは `GXcopy` が設定されています．「src」は源泉 RGB 値,「dst」は描画先ピクセルの RGB 値を示しています．

ファンクション名	論理演算
`GXclear`	0
`GXand`	src AND dst
`GXandReverse`	src AND NOT dst
`GXcopy`	src
`GXandInverted`	(NOT src) AND dst
`GXnoop`	dst
`GXxor`	src XOR dst
`GXor`	src OR dst
`GXnor`	(NOT src) AND (NOT dst)
`GXequiv`	(NOT src) XOR dst
`GXinvert`	NOT dst
`GXorReverse`	src OR (NOT dst)
`GXcopyInverted`	NOT src
`GXorInverted`	(NOT src) OR dst
`GXnand`	(NOT src) OR (NOT dst)
`GXset`	1

よく用いられる GC ファンクションの 1 つとして「`GXxor`」があります．源泉 RGB 値と描画先ピクセルの RGB 値との XOR をとった値を描画するので，2 回同じ場所を描画すると必ず元の値に戻ります．カーソル描画や領域表示を行なう場合によく用いられます．この他，画像の反転表示や，Red，Green，Blue の合成などでの利用が考えられます．

この関数で設定した GC ファンクションは各種描画関数と gputarea() 関数 (§2.4.41)，gputimage() 関数 (§2.4.42) で有効です．

なお，複数回同じ場所を描画した場合にそれぞれが同じ結果にならない GC ファンクションを指定した場合，drawsym() 関数 (§2.4.35)，drawsyms() 関数 (§2.4.36)，drawarrow() 関数 (§2.4.37) で描かれる図形についての結果は定義されていません．

使用例　`newgcfunction(win, GXxor) ;`

2.4.21　void pset(int wn, double x, double y)

機　能　点の描画

wn で指定したウィンドゥに点を描画します.

描画関数等で用いられるアプリケーション座標系は，デフォルトでは左下が (0.0, 0.0) で右上が (xsize-1.0, ysize-1.0) になっています．この座標は，coordinate() 関数 (§2.4.6) または window() 関数 (§2.4.7) で変更が可能です.

使用例　`pset(win,gx,gy) ;`

2.4.22　void drawline(int wn, double x0, double y0, double x1, double y1)

機　能　直線の描画

wn で指定したウィンドゥの (x0, y0) から (x1, y1) に直線を描画します.

使用例　`drawline(win,gx0,gy0,gx1,gy1) ;`

2.4.23　void moveto(int wn, double x, double y), void lineto(int wn, double x, double y)

機　能　連続的に直線を描く

lineto() 関数を複数回使う事により，wn で指定したウィンドゥに連続的に直線を描画します.

moveto() 関数は，(x, y) を lineto() 関数のための初期位置に設定します．lineto() 関数は，以前 `moveto()` または `lineto()` が呼ばれた時に指定された座標から，(x, y) へ直線を引きます．`moveto()` でペンを上げて移動，`lineto()` でペンを下ろして描画，と考えるとわかりやすいでしょう.

使用例　`lineto(win,gx,gy) ;`

2.4.24　void line(int wn, double x, double y, int mode)

機　能　連続的に直線を描く

この関数は，§2.4.23 の `moveto()`，`lineto()` と同じ動作をします.

新しいプログラムでは，moveto()・lineto() 関数を使用してください．

line() 関数を複数回使う事により，wn で指定したウィンドゥに連続的に直線を描画します．mode に PENDOWN を指定すると以前 line() 関数が呼ばれた時に指定された座標から，(x, y) へ直線を引きます．mode に PENUP を指定すると (x, y) を line() 関数の初期位置に設定します．mode=PENDOWN でペンを下ろして描画，mode=PENUP でペンを上げて移動と考えるとわかりやすいでしょう.

使用例　`line(win,gx,gy,PENDOWN) ;`

2.4.25　void drawpts(int wn, const double x[], const double y[], int n)

機　能　複数の点を描く

wn で指定したウィンドゥで，n 個の点を描きます．x，y は n 個の実数の一次元配列で，x[0]〜x[n-1]，y[0]〜y[n-1] に各点の座標を入れておきます.

引数 x，y が float 型の配列の場合も，double 型の場合と同じ関数名で使用可能です.

使用例　`drawpts(win,plx,ply,5) ;`

2.4.26　void drawlines(int wn, const double x[], const double y[], int n)

機　能　折れ線を描く

wn で指定したウィンドゥで，折れ線を描きます．x，y は n 個の実数の一次元配列で，x[0]〜x[n-1]，y[0]〜y[n-1] に折れ線の各点の座標を入れておきます.

引数 x，y が float 型の配列の場合も，double 型の場合と同じ関数名で使用可能です.

使用例　`drawlines(win,plx,ply,5) ;`

2.4.27　void drawpoly(int wn, const double x[], const double y[], int n)

機　能　多角形を描く

wn で指定したウィンドゥで，多角形を描きます．x，y は n 個の実数の一次元配列で，x[0]〜x[n-1]，y[0]〜y[n-1] に多角形の各点の座標を入れておきます.

引数 x，y が float 型の配列の場合も，double 型の場合と同じ関数名で使用可能です.

使用例　`drawpoly(win,plx,ply,5) ;`

2.4.28　void fillpoly(int wn, const double x[], const double y[], int n, int i)

機　能　多角形を塗り潰す

wn で指定したウィンドゥで，多角形の領域を塗り潰します．x，y は n 個の実数の一次元配列で，x[0]〜x[n-1]，y[0]〜y[n-1] に多角形の各点の座標を入れておきます．i は塗り潰す時の形状で通常は 0 を，凸多角形の時は 1 を指定します.

引数 x，y が float 型の配列の場合も，double 型の場合と同じ関数名で使用可能です.

使用例　`fillpoly(win,plx,ply,5,0) ;`

2.4.29　void drawrect(int wn, double x, double y, double w, double h)

機　能　長方形を描く

wn で指定したウィンドゥに，頂点 (x, y) から正の方向に幅 w，高さ h の長方形を描きます.

使用例　`drawrect(win,50.0,60.0,30.0,20.0) ;`

2.4.30　void fillrect(int wn, double x, double y, double w, double h)

機　能　長方形の領域を塗り潰す

wn で指定したウィンドゥで，頂点 (x, y) から正の方向に幅 w，高さ h の長方形の領域を塗り潰します.

使用例　`fillrect(win,50.0,60.0,30.0,20.0) ;`

2.4.31 void drawcirc(int wn, double xcen, double ycen, double xrad, double yrad)

機　能　中心座標，半径を与えて円を描く

wn で指定したウィンドゥに，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円を描きます．

別名として，circle が使えます．次の 2 つの使用例は，同じ動作となります．

使用例　`drawcirc(win,50.0,60.0,30.0,40.0) ;　 circle(win,50.0,60.0,30.0,40.0) ;`

2.4.32 void fillcirc(int wn, double xcen, double ycen, double xrad, double yrad)

機　能　中心座標，半径を与えて円を塗り潰す

wn で指定したウィンドゥに，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円を塗り潰します．

使用例　`fillcirc(win,50.0,60.0,30.0,40.0) ;`

2.4.33 void drawarc(int wn, double xcen, double ycen, double xrad, double yrad, double sang, double eang, int idir)

機　能　円の中心，半径，始点，終点の角度を与えて円弧を描く

wn で指定したウィンドゥに，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円弧を描きます．sang は開始角，eang は終了角で，度で与えます．idir は円弧を描く方向で 1 で左廻り，-1 で右廻りとなります．

使用例　`drawarc(win,50.0,60.0,30.0,40.0,-10.0,-170.0,-1) ;`

2.4.34 void fillarc(int wn, double xcen, double ycen, double xrad, double yrad, double sang, double eang, int idir)

機　能　円の中心，半径，始点，終点の角度を与えて円弧を塗り潰す

wn で指定したウィンドゥで，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad のの円弧を塗り潰します．sang は開始角，eang は終了角で，度で与えます．idir は円弧を描く方向で 1 で左廻り，-1 で右廻りとなります．

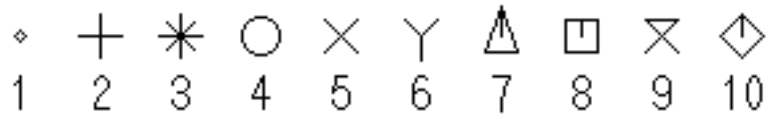
使用例　`fillarc(win,50.0,60.0,30.0,40.0,-10.0,-170.0,-1) ;`

2.4.35 void drawsym(int wn, int x, int y, int size, int sym_type)

機　能　センターシンボルの描画

wn で指定したウィンドゥにセンターシンボルを座標 (x, y) に描きます．size はシンボルの大きさでピクセル単位，sym_type でシンボルの種類を指定します．

sym_type とシンボルの関係は，次の図のとおりです．



使用例　`drawsym(win,gx,gy,16,2) ;`

2.4.36　void drawsyms(int wn, const double x[], const double y[], int n, int size, int sym_type)

機　能　複数のシンボルを描く

`wn` で指定したウィンドゥで，`n` 個のシンボルを描きます．`x`，`y` は `n` 個の実数の一次元配列で，`x[0]`〜`x[n-1]`，`y[0]`〜`y[n-1]` に各シンボルの座標を入れておきます．

`size` はシンボルの大きさでピクセル単位，`sym_type` でシンボルの種類を指定します．

`sym_type` とシンボルの関係については，§2.4.35をご覧ください．

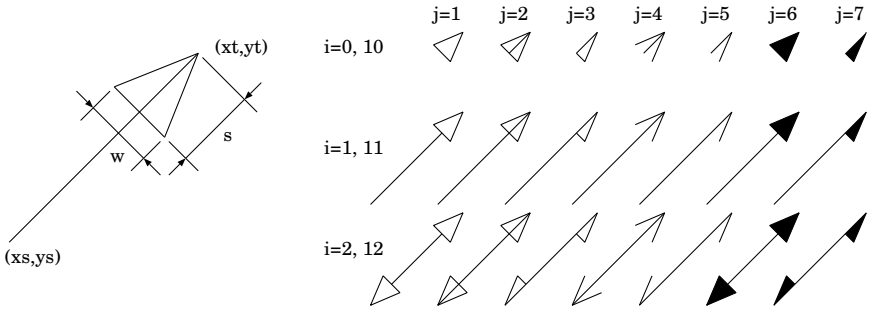
引数 `x`，`y` が float 型の配列の場合も，double 型の場合と同じ関数名で使用可能です．

使用例　`drawsyms(win,plx,ply,5,16,8) ;`

2.4.37　void drawarrow(int wn, double xs, double ys, double xt, double yt, double s, double w, int 10*i+j)

機　能　種々の型の矢印を描く

`wn` で指定したウィンドゥに (`xs`, `ys`) から (`xt`, `yt`) に向かって矢印を描きます．矢印の形状は以下の図の通りで，`s` と `w` は実数で指定します．`i` が `0`〜`2` の場合には `w`,`s` はピクセル数を，`i` が `10`〜`12` の場合には `w`,`s` は矢印の長さに対する割合で `0.0`〜`1.0` の値を指定します．



使用例　`drawarrow(win,gx0,gy0,gx1,gy1,0.3,0.2,114) ;`

2.4.38　int newfontset(int wn, const char *argsformat, ...)

機　能　フォントセット (日本語フォント) の指定

`wn` で指定したウィンドゥで描くフォントセットを指定します．文字の描画は drawstr() 関数を利用します．

`argsformat`(とそれに続く引数) で指定される文字列がフォントセットに設定されます．`argsformat` 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっているため，文字を拡大・縮小する時など，使用例のように引数を与えると大変便利です．

返り値は，指定されたフォントセットが取得できた場合は `0`，代替フォントで取得できた場合は正の値，フォントセットの取得に失敗した場合は負の値となります．

フォントセットの設定は，X サーバにインストールされたフォントを指定する必要があり，OS やディストリビューションに依存します．確実に表示したい場合は，次のような設定を推奨します:

14 ドットフォント	"-*-fixed-medium-r-normal--14-*"
16 ドットフォント	"-*-fixed-medium-r-normal--16-*"
24 ドットフォント	"-*-fixed-medium-r-normal--24-*"

使用例 `st = newfontset(win,"-kochi-gothic-medium-r-normal--%d-*-*-*-*-*-*-*",fsize) ;`

使用例
```
st = newfontset(win,"-adobe-helvetica-medium-r-*-*-12-*-*-*-*-*-iso8859-1,"
                    "-*-fixed-medium-r-normal--14-*") ;
```

2.4.39 int drawstr(int wn, double x, double y, int size, double theta, const char *argsformat, ...)

機　能　文字列の描画

wn で指定したウィンドゥに，文字列を座標 (x, y) から描きます．size は文字の大きさで，ピクセル単位で指定します．theta は文字列の回転を指定する引数ですが，現バージョンでは機能しません．argsformat(とそれに続く引数) で指定される文字列が描かれます．なお，argsformat 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっていますので，使用例のように変数の値などを描く事もできます．

文字のサイズ size は 1〜24 の範囲で指定できます．size と実際のフォントとの関係は以下の表のようになっています．この場合，文字は半角英数字のみ描画できます．

マルチバイト文字 (漢字) を描画する場合は，size には FONTSET を指定します．この場合のフォントの指定は，newfontset() 関数 (§2.4.38) を利用します．newfontset() でのフォント指定がない場合は，デフォルトの 14 ドットのフォントセットで描画されます．

この関数の返り値は描いた文字列の長さです．

1〜7 : 5×7	8 : 5×8	9 : 6×9	10〜11 : 6×10	12 : 6×12
13 : 7×13	14〜15 : 7×14	16〜19 : 8×16	20〜23 : 10×20	24 : 12×24

使用例 `drawstr(win,gx,gy,16,0,"velocity v=%f",v) ;`

使用例 `drawstr(win,gx,gy,FONTSET,0,"日本語の描画も OK!") ;`

2.4.40 void gscroll(int wn, int inc_x, int inc_y, int clr)

機　能　ピクセル単位で描画レイヤをスクロールする

wn で指定したウィンドゥの描画レイヤを，(inc_x, inc_y) ピクセル分スクロールさせます．引数 inc_x と inc_y はウィンドゥ座標系 (座標値は整数) での増分を与えます．

clr が 0 の場合，画面の上下左右がつながっている状態でスクロールしますが，clr が 1 の場合はスクロール・インした部分を背景色で初期化します．

次の例は，上方向に 2 ピクセルスクロールする例です．

使用例 `gscroll(win,0,2,1) ;`

なお，gsetinitialattributes() 関数 (§2.5.4) で，BOTTOM_LEFT_ORIGIN 属性が無効化されている場合は，この例では下方向にスクロールします．

2.4.41 void gputarea(int wn, double x, double y, int src_wn, int src_ly, double src_xs, double src_ys, double src_xe, double src_ye)

機　能　任意のウィンドゥ・レイヤ・領域の画像を描画レイヤにコピーする

ウィンドウ番号 src_wn，レイヤ番号 src_ly の (src_xs, src_ys) から (src_xe, src_ye) の範囲の画像を，ウィンドゥ番号 wn の座標 (x, y) にコピーします．座標 (x, y) には取り出した画像の原点を指定します．

デフォルトでは，src_wn から取り出した画像の原点は左下として扱われますが，gsetinitialattributes() 関数 (§2.5.4) で BOTTOM_LEFT_ORIGIN 属性が無効化されている場合は，画像の原点は左上として扱われます．

この関数の動作は，newgcfunction() 関数 (§2.4.20) による設定の影響を受けます．

使用例　`gputarea(win,gx,gy, wn1,0, 0.0,0.0,99.0,99.0) ;`

2.4.42 int gputimage(int wn, double x, double y, unsigned char *buf, int width, int height, int msk)

機　能　メモリバッファの (マスク付き) 画像データを描画レイヤに一括転送する (背景透過処理が可能)

wn で指定したウィンドゥの座標 (x, y) に buf に用意した，幅 width，高さ height の 24-bit 画像データを一括転送します．デフォルトでは，バッファに用意された画像の原点は左下として扱われますが，gsetinitialattributes() 関数 (§2.5.4) で BOTTOM_LEFT_ORIGIN 属性が無効化されている場合は，画像の原点は左上として扱われます．

この関数の動作は，newgcfunction() 関数 (§2.4.20) による設定の影響を受けます．

バッファbuf には，データをマスク値,Red,Green,Blue の順に 4 つずつ，水平方向に走査しながら画像の上から下へ用意します (buf[0]=0x0ff，buf[1]=red[0]，buf[2]=green[0]，buf[3]=blue[0]，といった具合いです)．Red,Green,Blue の値には 0x000〜0x0ff の範囲の輝度を与えて，マスク値は 0x0ff(不透明) か 0x000(透明) を与えます．

引数 msk にはマスク値の有無を与えます．マスク値を有効にする場合 (背景透過処理を有効にする場合) は 1 を，そうでない場合は 0 とします (これ以外の値は，将来アルファに対応した時に使う予定です)．

なお，X サーバの depth が 16 以上でない等のエラーの場合は，返り値として負の値を返します．正常終了の場合は 0 を返します．X サーバの depth は ggetdisplayinfo() 関数 (§2.5.1) で調べる事ができます．

EGGX/ProCALL のソースパッケージに収録している tools/ppmtoh.c または tools/xpmtoh.c[8] を利用すると，gputimage() 関数の第 4 引数から第 6 引数に与えるための値と配列を，ppm または xpm ファイルから作成する事ができます．次のようにして，ヘッダファイルを作成すると便利です．

```
$ ./ppmtoh my_image1.ppm >> my_images.h
$ ./xpmtoh my_image1.xpm >> my_images.h
```

使用例　`gputimage(win,gx,gy,buffer,640,400,1) ;`

2.4.43 unsigned char *ggetimage(int wn, int ly, double xs, double ys, double xe, double ye, int *r_width, int *r_height)

機　能　任意のウィンドゥ・レイヤ・領域の画像データをメモリバッファに取り込む

ウィンドウ番号 wn，レイヤ番号 ly の (xs, ys) から (xe, ye) の範囲の画像データをメモリバッファに取り込み，そのバッファのアドレスを返します．取り込まれた画像データの幅と高さは*r_width と*r_height に返ります．

[8] xpmtoh の利用には，netpbm のインストールが必要です．

メモリバッファには画像データが，0x0ff,Red,Green,Blue の順に 4 つずつ，水平方向に走査しながら画像の上から下へ格納されます．(buf[0]=0x0ff，buf[1]=red[0]，buf[2]=green[0]，buf[3]=blue[0]，といった具合いです)．Red,Green,Blue の値には 0x000〜0x0ff の範囲の輝度が入ります．

なお，**返されたバッファは，free() 関数を使ってユーザ・プログラム側で開放する必要があります**．

引数に設定された値が不正等のエラーの場合は，返り値として NULL を返します．

使用例　`buffer = ggetimage(win, 0, 0.0,0.0, 639.0,399.0, &width, &height) ;`

2.4.44　int gsaveimage(int wn, int ly, double xs, double ys, double xe, double ye, const char *conv, int nd, const char *argsformat, ...)

機　能　任意のウィンドゥ・レイヤ・領域の画像をコンバータ (netpbm 等) を通してファイルに保存するバックグラウンドプロセスを起動し，ウィンドウ番号 wn，レイヤ番号 ly の (xs, ys) から (xe, ye) の範囲をコンバータ (netpbm の各種コマンド,ImageMagick の convert 等) を通してファイルに保存します[9]．conv には ppm 形式から各画像フォーマットに変換するコンバータのコマンドラインを指定します．例えば，netpbm を利用して png 形式に保存する場合は，"pnmtopng"とします．ImageMagick を利用する場合は，"convert"とします．もちろん，オプションスイッチも含める事もでき，"pnmtops -scale 0.125" といった指定も可能です．次の引数 nd は減色パラメータで，R,G,B 1 チャンネルあたりの色の段階数を指定します．nd は簡単な図形では 16 くらいで十分ですが，多くの色を使っている場合は最大値の 256 を指定してください．

argsformat(とそれに続く引数) で指定される文字列がファイル名として扱われます．argsformat 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっているため，使用例のようにファイル名に変数の値を含む，といった事も可能です．このようにすると動画作成時に大変便利です．

ppm から他の画像フォーマットに変換する netpbm のコマンドの例を挙げておきます．各コンバータの使用方法に関しては端末から man コンバータ名 とタイプする事で調べることができます．ImageMagick の convert コマンドを利用する場合は，ファイル名のサフィックス (ドット以降の文字列，例えば.jpg，.png，.eps など) から保存フォーマットが決定されます．

画像フォーマット	コンバータ名
AutoCAD DXB	ppmtoacad
windows bitmap	ppmtobmp
Berkeley YUV	ppmtoeyuv
GIF	ppmtogif
NCSA ICR graphics	ppmtoicr
IFF ILBM	ppmtoilbm
Interleaf image	ppmtoleaf
HP LaserJet PCL 5 Color	ppmtolj
map	ppmtomap
Mitsubishi S340-10 printer	ppmtomitsu
Atari Neochrome .neo	ppmtoneo
PPC Paintbrush	ppmtopcx
portable graymap	ppmtopgm
Atari Degas .pi1	ppmtopi1
Macintosh PICT	ppmtopict
HP PaintJet	ppmtopj
HP PaintJet XL PCL	ppmtopjxl
X11 “puzzle”	ppmtopuzz
three portable graymaps	ppmtorgb3
DEC sixel	ppmtosixel
TrueVision Targa	ppmtotga

画像フォーマット	コンバータ名
Motif UIL icon	ppmtouil
Windows .ico	ppmtowinicon
XPM format	ppmtoxpm
Abekas YUV	ppmtoyuv
YUV triplets	ppmtoyuvsplit
DDIF	pnmtoddif
FIASCO compressed	pnmtofiasco
FITS	pnmtofits
JBIG	pnmtojbig
JFIF (”JPEG”) image	pnmtojpeg
Palm pixmap	pnmtopalm
plain (ASCII) anymap	pnmtoplainpnm
Portable Network Graphics	pnmtopng
PostScript	pnmtops
Sun raster	pnmtorast
RLE image	pnmtorle
sgi image	pnmtosgi
Solitaire image recorder	pnmtosir
TIFF file	pnmtotiff
CMYK encoded TIFF	pnmtotiffcmyk
X11 window dump	pnmtoxwd

[9] netpbm は http://sourceforge.net/projects/netpbm/ ，ImageMagick は http://www.imagemagick.org/ などで配布されています．

ここに挙げた以外のコンバータも，標準入力から ppm 形式で入力し，標準出力から変換データを出力するものであれば使用する事ができます.

コンバータを通さずに直接 ppm 形式で保存する場合は conv には""と指定してください．ただしその場合，非圧縮のバイナリデータが保存されるので，ファイルのサイズが大きくなります.

この関数では X サーバから画像データを転送し，保存するわけですが，ネットワークが遅かったりすると非常に時間がかかります．その事を考慮してこの関数では子プロセスを起動して，バックグラウンドで X サーバからの画像の転送・ディスクへの書き込みを行うようにしています．したがってユーザプログラムは gsaveimage(); の後すぐに他の作業ができるようになります．ただし，この子プロセスが終わるまでは X サーバでの描画等ができないので，もし gsaveimage(); のすぐ後に EGGX の描画関数などを呼んだ場合は画像の転送・保存が完了するまで待たされる事になります.

なお，gsaveimage を使っている場合は プログラムは必ず gclose() 関数 (§2.4.2) を呼んでから終了する ようにしてください.

返り値は，子プロセスの起動失敗などのエラーの場合は負の値，正常終了の場合は 0 となります.

使用例　gsaveimage(win,0, 0.0,0.0, 639.0,399.0, "pnmtopng",256,"img%d.png",i) ;
png 形式に保存する例です．gif 形式の場合は conv を"ppmtogif" にします.

使用例　gsaveimage(win,0, 0.0,0.0, 639.0,399.0,
"pnmtops -noturn -dpi 72 -equalpixels -psfilter -flate -ascii85",256,"figure.eps") ;
PostScript 形式に保存する例です．netpbm の pnmtops では RunLength 圧縮や GZIP 圧縮 (可逆圧縮) をサポートしています．上記のように「-psfilter -flate -ascii85」をつける事で画質を損うことなくファイルサイズを小さくできます.

2.4.45 unsigned char *readimage(const char *conv, const char *filename, int *r_width, int *r_height, int *r_msk)

機　能　ファイルの画像データをコンバータ (netpbm 等) を通してメモリバッファに取り込む

ファイル filename の画像データをコンバータ (netpbm の各種コマンド,ImageMagick の convert 等) を通してメモリバッファに読み込み，そのバッファのアドレスを返します．取り込まれた画像データの幅と高さが，*r_width と*r_height に返ります．アルファ値が読み込まれた場合には，*r_msk に 1 以上の値が返り，そうでない場合には 0 が返ります．1 が返るのは，アルファ値が 0x000，0x0ff の二値だった場合です.

引数 conv には，各画像フォーマットから pbm,pgm,ppm,pam のいずれかの形式に変換するコンバータのコマンドラインを指定します．例えば，netpbm を利用して png 形式のファイルを読み取る場合は，"pngtopnm"(新しいバージョンでは"pngtopam") とします．ImageMagick を利用する場合は，"convert" とします．netpbm，ImageMagick 以外のコンバータも，標準入力から filename の内容を入力し，標準出力から変換データを出力するものであれば使用する事ができます．もちろん，conv にはオプションスイッチも含める事もできます.

コンバータを通さずに直接ファイルを読み込む場合は conv には""と指定してください．ただし，読み込めるフォーマットはバイナリ形式の pbm,pgm,ppm,pam のいずれかに限定されます.

メモリバッファには画像データが，Alpha(マスク値),Red,Green,Blue の順に 4 つずつ，水平方向に走査しながら画像の上から下へ格納されます．(buf[0]=alpha[0], buf[1]=red[0], buf[2]=green[0], buf[3]=blue[0], といった具合いです)．Alpha,Red,Green,Blue の値には 0x000〜0x0ff の範囲の輝度が入ります.

なお，**返されたバッファは，free() 関数を使ってユーザ・プログラム側で開放する必要があります**.

ファイルが読み込めない等のエラーが発生した場合は，返り値として NULL を返します.

使用例　buffer = readimage("pngtopnm", "myimage.png", &width, &height, &msk) ;

2.4.46 int writeimage(const unsigned char *buf, int width, int height, int msk, const char *conv, int nd, const char *argsformat, ...)

機　能　メモリバッファの画像データをコンバータ (netpbm 等) を通してファイルに保存する

引数 buf で指定されたメモリバッファに格納された，高さ width，幅 height の画像データをコンバータ (netpbm の各種コマンド,ImageMagick の convert 等) を通してファイルに保存します．引数 msk には，アルファ値 (マスク) を保存する場合は 1 以上の値を，そうでない場合は 0 を指定します．conv には ppm または pam 形式から各画像フォーマットに変換するコンバータのコマンドラインを指定します．例えば，netpbm を利用して png 形式に保存する場合は，"pnmtopng"(アルファなし) または"pamrgbatopng"(アルファあり) とします．ImageMagick を利用する場合は，"convert"とします．もちろん，オプションスイッチも含める事もでき，"pnmtops -scale 0.125" といった指定も可能です．次の引数 nd は減色パラメータで，R,G,B 1 チャンネルあたりの色の段階数を指定します．nd は簡単な図形では 16 くらいで十分ですが，多くの色を使っている場合は最大値の 256 を指定してください．

argsformat(とそれに続く引数) で指定される文字列がファイル名として扱われます．argsformat 以降は，C 標準関数の printf() 関数の場合と同様の可変引数となっているため，使用例のようにファイル名に変数の値を含む，といった事も可能です．このようにすると動画作成時に大変便利です．

バッファbuf には，データを Alpha(マスク値),Red,Green,Blue の順に 4 つずつ，水平方向に走査しながら画像の上から下へ用意します (buf[0]=0x0ff，buf[1]=red[0]，buf[2]=green[0]，buf[3]=blue[0]，といった具合いです)．Alpha,Red,Green,Blue の値には 0x000〜0x0ff の範囲の輝度を与えます．

ImageMagick の convert コマンドを利用する場合は，ファイル名のサフィックス (ドット以降の文字列，例えば.jpg，.png，.eps など) から保存フォーマットが決定されます．

netpbm，ImageMagick 以外のコンバータも，標準入力から ppm または pam 形式で入力し，標準出力から変換データを出力するものであれば使用する事ができます．

コンバータを通さずに直接 ppm または pam 形式で保存する場合は conv には""と指定してください．ただしその場合，非圧縮のバイナリデータが保存されるので，ファイルのサイズが大きくなります．

コンバータあるいはファイルに対して，ppm と pam のどちらが出力されるかは，引数 msk とファイル名のサフィックスで決まります．pam が出力されるのは,「ファイル名のサフィックスが".pam"の場合」「msk が 1 以上の場合」です．それ以外の場合は ppm が出力されます．

返り値は，子プロセスの起動失敗などのエラーの場合は負の値，正常終了の場合は 0 となります．

コンバータのコマンドの詳細については，§2.4.44をご覧ください．

使用例　`writeimage(buffer,640,400,0,"pnmtopng",256,"img%d.png",i) ;`

2.4.47 void gsetnonblock(int flag)

機　能　ggetch()，ggetevent()，ggetxpress() の動作モードを設定する

デフォルトでは，キーボードやマウスの入力情報を取得する関数ggetch()，ggetevent()，ggetxpress() が呼ばれると，入力があるまで関数から抜けずに待ち続けます (ブロッキングモード)．

flag に ENABLE を設定して gsetnonblock() 関数が呼ばれるとノンブロッキングモードとなり，ggetch()，ggetevent()，ggetxpress() は入力の有無にかかわらずすぐに関数から戻るようになります．

デフォルトのブロッキングモードに戻すには，flag に DISABLE を与えてください．

gsetnonblock() 関数の呼び出しは，ウィンドゥのオープンの前あるいは後のどのタイミングでも有効です．

使用例　`gsetnonblock(ENABLE) ;`

2.4.48 int ggetch()

機 能 キーボードから入力された文字を返す

EGGX で開いたすべてのウィンドゥからのキーボードの入力情報を返します．ブロッキングモード (デフォルト) ではキー入力があるまで待ち続けますが，ノンブロッキングモードでは入力の有無にかかわらずすぐに関数から戻ります (動作モードについては §2.4.47の gsetnonblock() 関数を参照してください)．ノンブロッキングモードにおいて入力が無かった場合は，負の値が返ります．

ggetch() 関数は，端末からの 1 文字入力 `fgetc(stdin)` と似ていますが，改行まで待たないという点と，0x001〜0x01f，0x07f の「特殊キー」と「Ctrl+アルファベットキー」の入力を拾う点が異っています．

次の表で 16 進の文字コードを知る事ができます．イタリック体の数字が 16 進 2 桁の上位を表します．

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0		Home	PageUp	Pause		End	PageDown		BackSpace	Tab				Enter		
1												Esc	→	←	↑	↓
2	Space	!	”	#	$	%	&	’	(	)	∗	+	,	−	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	＼	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	Delete

例えば,「a」は 0x061,「A」は 0x041 です．

0x001〜0x01a の中にはいくつか特殊キーが含まれていますが，この区間は「Ctrl+アルファベットキー」のコードと共有となります．例えば,「BackSpace」と「Ctrl+H」のコードはどちらも 0x008 となります．0x001〜0x01a で空白になっている個所は,「Ctrl+アルファベットキー」にのみコードが割り当てられています．

キーボード入力があったウィンドゥ番号をチェックしたい場合は，ggetxpress() 関数 (§2.4.50) を使ってください．

使用例
```
key=ggetch() ;
```

2.4.49 int ggetevent(int *type, int *button, double *x, double *y)

機 能 マウスやキーボードからの入力の情報を返す

EGGX で開いたすべてのウィンドゥからのマウスやキーボードの入力情報を返します．ブロッキングモード (デフォルト) では入力があるまで待ち続けますが，ノンブロッキングモードでは入力の有無にかかわらずすぐに関数から戻ります (動作モードについては §2.4.47の gsetnonblock() 関数を参照してください)．ノンブロッキングモードにおいて入力が無かった場合は，負の値が返ります．

この関数の返り値は，入力のあったウィンドゥ番号です．これを使って，ユーザ・プログラム側で意図したウィンドゥからの入力かどうかをチェックします．

`*type` には，マウスのモーションの場合は `MotionNotify`，`EnterNotify`，`LeaveNotify` のいずれか，マウスのボタンが押された場合は `ButtonPress`，キーボードからの入力の場合は `KeyPress` が返ります．`EnterNotify`，`LeaveNotify` は，それぞれマウスカーソルがウィンドゥに入った時，ウィンドゥから出た時に発行されます．

マウスの入力の場合，`*button` にはクリック (on･off) された，または押されているボタンの番号 (`1,2,3,`…) が代入され，ウィンドゥ上でのマウスポインタの座標 (アプリケーション座標系) が`*x`，`*y` に代入されます．

キー入力の場合は，キーコードが`*button` に返ります．キーコードは ggetch() 関数 (§2.4.48) の返り値と同一です．

4 つの引数 type，button，x，y それぞれについて値を取り出す必要がない場合は，NULL を与えてもかまいません．なお，C++のコードでは，NULL は必ず (double *) または (float *) でキャストしてください．

引数 x，y が float 型のポインタ変数の場合も，double 型の場合と同じ関数名で使用可能です．

使用例　`win_ev=ggetevent(&type,&b,&x,&y) ;`

2.4.50　int ggetxpress(int *type, int *button, double *x, double *y)

機　能　マウスからのボタンクリック，キーボードからの入力の情報を返す

EGGX で開いたすべてのウィンドゥからのマウスのボタンクリック，キータイプの入力情報を返します．この関数は，ggetevent() 関数 (§2.4.49) の簡易版です．ブロッキングモード (デフォルト) では入力があるまで待ち続けますが，ノンブロッキングモードでは入力の有無にかかわらずすぐに関数から戻ります (動作モードについては §2.4.47の gsetnonblock() 関数を参照してください)．ノンブロッキングモードにおいて入力が無かった場合は，負の値が返ります．

この関数の返り値は，入力のあったウィンドゥ番号です．これを使って，ユーザ・プログラム側で意図したウィンドゥからの入力かどうかをチェックします．

*type には，マウスのボタンクリックの場合は ButtonPress，キーボードからの入力の場合は KeyPress が返ります．

マウスのボタンクリックの場合，*button にはクリックされたボタンの番号 (1,2,3,…)，クリックされた時のウィンドゥ上でのマウスポインタの座標 (アプリケーション座標系) が*x，*y に代入されます．

キー入力の場合は，キーコードが*button に返ります．キーコードは ggetch() 関数 (§2.4.48) の返り値と同一です．

4 つの引数 type，button，x，y それぞれについて値を取り出す必要がない場合は，NULL を与えてもかまいません．なお，C++のコードでは，NULL は必ず (double *) または (float *) でキャストしてください．

引数 x，y が float 型のポインタ変数の場合も，double 型の場合と同じ関数名で使用可能です．

使用例　`win_ev=ggetxpress(&type,&b,&x,&y) ;`

2.4.51　void msleep(unsigned long msec)

機　能　ミリ秒単位で実行を延期する

少なくとも msec ミリ秒の間，プログラムの実行をなにもせずに待ちます．アニメーション速度の調整に利用できます．

一般的には，時間の精度は 10 ミリ秒の単位です．

使用例　`msleep(100) ;`

2.5　EGGX の Advanced 関数リファレンス

以下でとりあげる関数は，より細かい制御を行なったりするためのもので，C 言語に習熟していないと利用が難しい関数もあります．なお，これらの関数のほとんどは，それぞれに対応する FORTRAN のルーチンは用意していません．

2.5.1 int ggetdisplayinfo(int *depth, int *root_width, int *root_height)

機　能　X サーバの情報 (depth，画面サイズ) を取得する

X サーバに接続し，depth，画面サイズを調べます．*depth は EGGX でウィンドゥをオープンした時に使われる depth 値 (ピクセルに何ビット使っているか)，すなわち 8(PseudoColor：256 色カラー)，16(TrueColor：65536 色カラー)，24(TrueColor：1677 万色カラー) といった値が代入されます．*root_width, *root_height はそれぞれ X サーバの画面のピクセルサイズが代入されます．

3 つの引数 depth，root_width, root_height それぞれについて値を取り出す必要がない場合は，NULL を与えてもかまいません．

関数の返り値は，X サーバへの接続が成功した場合は 0，失敗した場合は負の値となります．

使用例　`status = ggetdisplayinfo(&depth,NULL,NULL) ;`

2.5.2 void gsetnonflush(int flag)，int ggetnonflush()

機　能　描画関数等におけるフラッシュに関する設定

デフォルトでは，EGGX の描画関数やウィンドゥの装飾に関する関数は，関数から抜ける直前に XFlush() を呼んで，直ちに X サーバが EGGX からの命令を反映するようにしています (自動フラッシュモード)．しかし，X サーバによっては，XFlush() の連発によって描画パフォーマンスが低下する事があります．

flag に ENABLE を設定して gsetnonflush() 関数が呼ばれると，それ以降，EGGX の関数内部で XFlush() が呼ばれなくなり，ユーザ・プログラム側で，gflush() 関数 (§2.5.3) を呼んで任意のタイミングでフラッシュする事ができるようになります (非自動フラッシュモード)．

デフォルトの状態に戻すには，flag に DISABLE を与えてください．

gsetnonflush() 関数の呼び出しは，ウィンドゥのオープンの前あるいは後のどのタイミングでも有効です．

ggetnonflush() 関数は，現在の自動フラッシュに関する設定を読み取ります．gsetnonflush() 関数で設定された値が返ります．

使用例　`gsetnonflush(ENABLE) ;`

2.5.3 void gflush()

機　能　描画命令等をフラッシュする

描画関数等で発行した X サーバに対する一連の命令をフラッシュします．gsetnonflush() 関数で非自動フラッシュモードを設定した時に使います．

使用例　`gflush() ;`

2.5.4 void gsetinitialattributes(int values, int att_msk)

機　能　gopen() でのウィンドゥ属性を変更する

gopen() 関数で開かれる新規ウィンドゥの各種属性を変更します．gsetinitialattributes() 関数で一旦ある属性を変更すると，以降 gopen() 関数で開くウィンドゥすべてにその指定が反映されます．

ウィンドゥ属性は，次に示す 4 つのビットフラグで表現され，values には，ENABLE(全有効)，DISABLE(全無効) あるいは任意の属性値を指定し，att_msk には各属性に対する変更フラグ (マスク) を指定します．

- BOTTOM_LEFT_ORIGIN

ウィンドゥの左下をウィンドゥ座標系の原点 (0,0) にします．デフォルトではこの属性は有効になっています.

この属性を無効に設定すると，ウィンドゥ座標系の原点はウィンドゥの左上になります.

- SCROLLBAR_INTERFACE

  EGGX によるスクロールバー・インタフェースを提供します．デフォルトではこの属性は有効になっています.

  この属性を無効に設定すると，ウィンドゥがリサイズされてもスクロールバーはつきません.

- MAX_WINDOW_SIZE

  グラフィックス領域の大きさを越えたウィンドゥサイズへのリサイズを禁止します。デフォルトではこの属性は無効になっています.

- OVERRIDE_REDIRECT

  X における OverrideRedirect 属性を指定します．デフォルトではこの属性は無効になっています.

  OverrideRedirect 属性で開いたウィンドゥは，ウィンドゥマネージャの介入をうけず，ウィンドゥに枠がつきません．また，ウィンドゥを開いた時は常に最前面に表示されます．このモードはアプリケーション起動時のバナー表示などに用いられる事があります.

- DOCK_APPLICATION

  この属性を有効にすると，AfterStep，WindowMaker のアプレットとなるための各種設定を行なうようになります．デフォルトではこの属性は無効になっています.

  この属性を有効に設定すると，ウィンドゥがリサイズされてもスクロールバーはつきません.

次の例は，DOCK_APPLICATION 属性を有効にする例です.

使用例　`gsetinitialattributes(ENABLE, DOCK_APPLICATION) ;`

複数の属性を変更する場合は，次のようにします．この例では，OVERRIDE_REDIRECT 属性を有効にし，BOTTOM_LEFT_ORIGIN 属性を無効にしています.

使用例　`gsetinitialattributes(OVERRIDE_REDIRECT, BOTTOM_LEFT_ORIGIN | OVERRIDE_REDIRECT) ;`

属性値のすべてのビットを設定する場合は，次のように att_msk を-1 に (すなわち全ビットを 1 に) します.

使用例　`gsetinitialattributes(SCROLLBAR_INTERFACE | BOTTOM_LEFT_ORIGIN, -1) ;`

2.5.5 int ggetinitialattributes()

機　能　gopen() でのウィンドゥ属性を取得する

現在設定されている，gopen() のための属性値を読み取ります．gsetinitialattributes() 関数で設定された値が返ります.

使用例　`att=ggetinitialattributes() ;`

2.5.6 void gsetinitialbgcolor(const char *argsformat, ...)

機 能 gopen() での背景カラーを指定する

gopen() 関数で開かれる新規ウィンドゥの背景色 (gclr() 関数 (§2.4.11) で初期化される色) を指定します. argsformat(とそれに続く引数) で指定される文字列を背景色として設定します. 2 つめの引数 argsformat 以降は, C 標準関数の printf() 関数の場合と同様の可変引数となっています. この背景色の文字列には, X サーバの rgb.txt[10] に設定されている色か, "#c0c0ff"のように, 16 進数の Red,Green,Blue を指定します.

argsformat が NULL の場合は, デフォルトの設定 (黒) に戻します.

使用例 gsetinitialbgcolor("white") ;

2.5.7 void gsetborder(int wn, int width, const char *argsformat, ...)

機 能 ウィンドゥのホーダーと色を指定する

wn で指定されたウィンドゥのボーダー (枠) の幅ととカラーを変更します. 引数 width にはボーダーの幅をピクセル単位で指定します. argsformat(とそれに続く引数) で指定される文字列をボーダーの色に設定します. 3 つめの引数 argsformat 以降は, C 標準関数の printf() 関数の場合と同様の可変引数となっています.

一般的に, ウィンドゥのボーダーはウィンドゥマネージャによって再設定されるので, ここで設定した値は無効になります. ただし, OverrideRedirect 属性なウィンドゥの場合は, この設定が反映されます. OverrideRedirect 属性については, gsetinitialattributes() 関数 (§2.5.4) の解説を参照してください.

width が負の場合, argsformat が NULL の場合はそれぞれ設定変更を行ないません.

使用例 gsetborder(win,1,"white") ;

2.5.8 void gsetinitialborder(int width, const char *argsformat, ...)

機 能 gopen() でのウィンドゥのボーダーを指定する

gopen() 関数で開かれる新規ウィンドゥのボーダー (枠) の幅とカラーを指定します. 引数 width にはボーダーの幅をピクセル単位で指定します. argsformat(とそれに続く引数) で指定される文字列をボーダーの色に設定します. 2 つめの引数 argsformat 以降は, C 標準関数の printf() 関数の場合と同様の可変引数となっています.

width が負の場合, argsformat が NULL の場合はそれぞれ設定変更を行ないません.

デフォルトではボーダー幅は 0, ボーダーカラーは Black に設定されています.

使用例 gsetinitialborder(1,"White") ;

2.5.9 void gsetinitialgeometry(const char *argsformat, ...)

機 能 x,y の値を含む文字列から gopen() でのウィンドゥの大きさと出現座標を設定する

gopen() 関数で開かれる新規ウィンドゥの大きさと出現位置を, argsformat(とそれに続く引数) で指定される文字列から決定します. 2 つめの引数 argsformat 以降は, C 標準関数の printf() 関数の場合と同様の可変引数となっています. X11 のクライアントでは標準的な「-geometry」に続くコマンドオプション, 例えば "800x600+100-200" ような文字列を与えることができます.

[10] rgb.txt は UNIX 系の OS なら /usr/X11R6/lib/X11/ などにあります.

argsformat が NULL の場合にはデフォルトすなわち無指定の状態に戻します.
使用例のようにすると，整数値を与えることもできます.

使用例　gsetinitialgeometry("800x600%+d%+d",-30,40) ;

2.5.10　void gsetinitialwinname(const char *storename, const char *iconname, const char *resname, const char *classname)

機　能　gopen() でのウィンドゥ名，アイコン名，リソース名，クラス名を指定する
デフォルトでは，ウィンドゥ名やアイコン名はユーザプログラムのコマンド名から決定されますが，この関数を使う事で gopen() 関数で開かれる新規ウィンドゥのウィンドゥ名，アイコン名，リソース名，クラス名を引数 storename，iconname，resname，classname で指定できます.
それぞれの引数に NULL を与えると，それぞれをデフォルトの設定に戻します.

使用例　gsetinitialwinname("AyuClock","AyuClock","ayuclock","AyuClock") ;

2.5.11　void gsetscrollbarkeymask(int wn, unsigned int keymask)

機　能　EGGX によるスクロールバーのキー操作のためのキーマスクを設定する
EGGX では，ディスプレイのサイズに収まりきらない画素サイズを引数として gopen() 関数が呼ばれた場合や，ユーザがウィンドゥをリサイズした場合に，スクロールバー・インタフェースを提供します.
このスクロールバーは，デフォルトでは Alt キーを押しながらカーソルキー等で操作できるようになっていますが，この関数でこの仕様を変更できます.
引数 keymask には，次の表にある値を与えます.

マスク値	動作
ShiftMask	Shift キーが押されている時のみスクロールーバーのキー操作が有効
LockMask	CapsLock されている時のみスクロールーバーのキー操作が有効
ControlMask	Ctrl キーが押されている時のみスクロールーバーのキー操作が有効
Mod1Mask	Alt キーが押されている時のみスクロールーバーのキー操作が有効
Mod2Mask	NumLock されている時のみスクロールーバーのキー操作が有効
Mod5Mask	ScrollLock されている時のみスクロールーバーのキー操作が有効
0	常にスクロールーバーのキー操作が有効

keymask に 0 を与えた場合には，ggetch() 関数 (§2.4.48) 等でカーソルキー等の入力イベントを扱う事ができなくなりますので，注意してください.

使用例　gsetscrollbarkeymask(win, ShiftMask) ;

2.5.12　int generatecolor(color_prms *p, double dmin, double dmax, double data, int *r, int *g, int *b)

機　能　変数からカラーを生成する (コントラスト，ブライトネス，γ 補正などが可能)
変数 data の値を使って，r,g,b それぞれに 256 段階の Red,Greed,Blue のカラーを生成します．変数の最小，最大は dmin,dmax で指定します．この関数で得た，r,g,b の値は newrgbcolor() 関数 (§2.4.15) にそのまま与えて使う事ができます.
返り値は data が dmin,dmax の範囲内にあった場合は 0 を返し，dmin 未満であれば負の値を，dmax を越えていれば，正の値を返します.

現在の EGGX では，構造体 color_prms は次のようなメンバ構成になっていますが，将来拡張する可能性がありますので，変数宣言時に初期値を代入するコードは書かない事をお勧めします.

```
typedef struct _color_prms {
    int colormode ;
    int flags ;
    double contrast ;
    double brightness ;
    double gamma ;
    int seplevel ;
    void *ptr ;
    void (*function)( double,void *,double,double,double,double *,double *,double * ) ;
} color_prms ;
```

colormode はカラーパターンの番号で，詳細は，makecolor() 関数 (§2.4.17) をご覧ください.

flags は，コントラスト contrast，ブライトネス brightness，ガンマ補正 gamma，カラーセパレーションレベル seplevel，ユーザ関数 function，のどれを有効にするかを示すためのフラグです.

フラグ	対応するメンバ	解説
C_REVERSE	-	カラーパターンをデータ data の大小と逆にする
CP_CONTRAST	contrast	コントラスト制御を有効化．($0 \leq$ contrast ≤ 1)
CP_BRIGHTNESS	brightness	ブライトネス制御を有効化．($0 \leq$ brightness ≤ 1)
CP_GAMMA	gamma	ガンマ補正を有効化．($0 \leq$ gamma ≤ 1)
CP_SEPLEVEL	seplevel	カラーセパレーションレベルを有効化．($2 \leq$ seplevel)
CP_FUNCTION	function	ユーザ関数呼出しを有効化.

seplevel を使うと，リニアに変化するカラーをデジタイズできます．例えば，DS9_GRAY8 ではリニアに黒から白へ変化しますが，seplevel=10 に設定しておくと，10 色を使って段階的に変化するようになります.

function は generatecolor() が処理の最後に実行するユーザ関数です．function の引数はそれぞれ，data を Max=1.0,Min=0.0 で標準化した値，ptr，Red,Green,Blue の値 (in と out) となっています．この部分は EGGX のソースを見て理解できる方のみご利用ください.

使用例

```
color_prms cl ;
cl.colormode  = DS9_RAINBOW ;
cl.flags      = CP_CONTRAST | CP_BRIGHTNESS | CP_GAMMA ;
cl.contrast   = 1.0 ;
cl.brightness = 0.0 ;
cl.gamma      = 1.0 ;
    :
generatecolor(&cl,zmin,zmax,zvalue,&cl_r,&cl_g,&cl_b) ;
```

3 FORTRAN 編

3.1 ProCALL のルーチン一覧

3.1.1 ProCALL 標準ルーチン

章	ルーチン名	機能
§3.4.1	ggetdisplayinfo	X サーバの情報 (depth, 画面サイズ) を取得する
§3.4.2	gopen	任意のサイズのグラフィックス用ウィンドゥを開く
§3.4.3	gclose	任意のグラフィックス用ウィンドゥを閉じる
§3.4.4	gcloseall	すべてのグラフィックス用ウィンドゥを閉じ，X サーバと接続を断つ
§3.4.5	newcoordinate	アプリケーション座標系の変更 (参照点の座標とスケールを与える)
§3.4.6	newwindow	アプリケーション座標系の変更 (左下と右上の座標を与える)
§3.4.7	layer	レイヤの設定を行う
§3.4.8	copylayer	レイヤのコピーを行う
§3.4.9	gsetbgcolor	ウィンドゥのバックグラウンドカラー (gclr での色) を指定する
§3.4.10	gclr	グラフィックス用ウィンドゥの消去
§3.4.11	tclr	端末画面の消去
§3.4.12	newpencolor	描画色 (16 色) の変更
§3.4.13	newcolor	描画色の変更 (X サーバの持つ色を直接指定)
§3.4.14	newrgbcolor	描画色の変更 (Red,Green,Blue の輝度を指定)
§3.4.15	newhsvcolor	描画色の変更 (Hue,Saturation,Value を指定)
§3.4.16	makecolor	変数からカラーを生成する (カラーバー生成)
§3.4.17	newlinewidth	線幅の変更
§3.4.18	newlinestyle	線のスタイルの変更
§3.4.19	pset	点の描画
§3.4.20	drawline	直線の描画
§3.4.21	moveto, lineto	連続的に直線を描く
§3.4.23	drawpts	複数の点を描く
§3.4.24	drawlines	折れ線を描く
§3.4.25	drawpoly	多角形を描く
§3.4.26	fillpoly	多角形を塗り潰す
§3.4.27	drawrect	長方形を描く
§3.4.28	fillrect	長方形の領域を塗り潰す
§3.4.29	drawcirc	中心座標と半径を与えて円を描く
§3.4.30	fillcirc	中心座標と半径を与えて円を塗り潰す
§3.4.31	drawarc	円の中心，半径，始点，終点の角度を与えて円弧を描く
§3.4.32	fillarc	円の中心，半径，始点，終点の角度を与えて円弧を塗り潰す
§3.4.33	drawsym	1 個のシンボルの描画
§3.4.34	drawsyms	複数のシンボルを描く
§3.4.35	drawarrow	種々の矢印の描画
§3.4.36	newfontset	フォントセット (日本語フォント) の指定
§3.4.37	drawstr	文字列の描画
§3.4.38	drawnum	実数の値を描く
§3.4.39	putimg24	整数配列に用意した画像をウィンドゥに一括転送する
§3.4.40	saveimg	特定のレイヤの画像をコンバータ (netpbm 等) を通してファイルに保存する
§3.4.41	gsetnonblock	ggetch，ggetevent，ggetxpress ルーチンの動作モードを設定する
§3.4.42	ggetch	キーボードから入力された文字を返す
§3.4.43	ggetevent	マウスやキーボードからの入力の情報を返す
§3.4.44	ggetxpress	マウスからのボタンクリック，キーボードからの入力の情報を返す
§3.4.45	selwin	カルコンプ互換ルーチンで，どのウィンドゥにアクセスするか指定する

3.1.2 カルコンプ互換ルーチン

章	ルーチン名	機能
§3.5.1	plots	640 × 400 ピクセルのグラフィックス用ウィンドゥを開く
§3.5.2	window	座標系の変更
§3.5.3	newpen	描画色の変更
§3.5.4	clsc	端末画面の消去
§3.5.5	clsx	グラフィックス画面の消去
§3.5.6	plot	直線・点の描画
§3.5.7	arc	円の中心，半径，始点，終点の角度を与えて円弧を描く
§3.5.8	circ1	中心座標と半径を与えて円を描く
§3.5.9	arohd	種々の矢印の描画
§3.5.10	symbol	文字列または 1 個のシンボルの描画
§3.5.11	number	実数の値を描く

3.1.3 補助ルーチン

章	ルーチン名	機能
§3.6.1	msleep	ミリ秒単位で実行を延期する
§3.6.2	isnan	実数型変数が非数 (Not a Number) かどうか調べる
§3.6.3	rtoc	実数型変数を文字列に変換する

3.2 基本的な使用方法

ユーザプログラムについての特別な注意はありません．プログラムを書いたら，コンパイルは `egg` コマンドを用います．

例　`egg program.f`

3.3 描画速度をアップする方法

layer ルーチン (§3.4.7) と copylayer ルーチン (§3.4.8) とを使って，描画ルーチンでは常に非表示レイヤに描きましょう．描き終わったら，copylayer ルーチンを使って，非表示レイヤを表示レイヤにコピーします．このようにする事で，かなり描画速度が改善されます．

3.4 ProCALL 標準ルーチンのリファレンス

3.4.1 ggetdisplayinfo(ndepth,nrwidth,nrheight)

機　能　X サーバの情報 (depth，画面サイズ) を取得する

X サーバに接続し，depth，画面サイズを調べます．`ndepth` には ProCALL でウィンドゥをオープンした時に使われる depth 値 (ピクセルに何ビット使っているか)，すなわち `8`(PseudoColor：256 色カラー)，`16`(TrueColor：65536 色カラー)，`24`(TrueColor：1677 万色カラー) といった値が返ってきます．`nrwidth`，`nrheight` には，それぞれ X サーバの画面のピクセルサイズが返ります．なお，X サーバへの接続が失敗した場合は `ndepth` には負の値が返ってきます．

使用例　`call ggetdisplayinfo(ndepth,nrwidth,nrheight)`

3.4.2 gopen(nxsize,nysize,nw)

機 能 任意のサイズのグラフィックス画面を開く

任意のサイズの描画領域を持つウィンドゥを開きます.

引数 nxsize，nysize にはそれぞれ横方向，縦方向の描画領域のピクセル数を指定します．指定できる最大値は，32767 です.

nw には ProCALL で使用するウィンドゥ番号 (整数) が返ってきます．gopen を呼ぶと，nw には ProCALL 側で生成されたウィンドゥ番号が設定されるため，**ユーザが nw に値を代入しておく必要はありません**．ここで取得したウィンドゥ番号は ProCALL のグラフィックス描画ルーチンに渡します.

デフォルトでは，ルートウィンドゥ(つまり背景) のピクセル数とほぼ同じ，あるいはそれを越えるサイズの描画領域が指定された場合には，描画領域よりも小さいウィンドゥを開きます[11)]．この場合，スクロールバー・インタフェースが提供され，マウスまたはキーボードから任意の描画領域を表示する事が可能です.

互換ルーチンでどのウィンドゥにアクセスするかは，このウィンドゥ番号 nw を selwin ルーチン (§3.4.45) で指定します.

使用例 `call gopen(800,600,nwin)`

3.4.3 gclose(nw)

機 能 グラフィックス用ウィンドゥを閉じる

nw で指定されたウィンドゥを閉じます.

使用例 `call gclose(nwin)`

3.4.4 gcloseall

機 能 すべてのグラフィックス用ウィンドゥを閉じ，X サーバと接続を断つ

全てのウィンドゥを閉じ，X サーバと接続を断ち，ライブラリの内部処理で利用しているメモリ領域を開放します.

使用例 `call gcloseall`

3.4.5 newcoordinate(nw,nxw,nyw,xa,ya,xscale,yscale)

機 能 アプリケーション座標系の変更 (参照点の座標とスケールを与える)

nw で指定したウィンドゥのアプリケーション座標系を変更します

ウィンドゥ上のウィンドゥ座標系 (座標値は整数) は，左下が (0, 0) で，右上が (nxsize-1, nysize-1) であり，デフォルトではアプリケーション座標系 (座標値は実数) の座標値はウィンドゥ座標系のそれに一致しています.

newcoordinate ルーチンを使う事により，アプリケーション座標系の (xa, ya) をウィンドゥ座標系の (nxw, nyw) に対応させ，それぞれのスケーリングファクターを xscale，yscale で指定します．すなわち，描画ルーチン等におけるアプリケーション座標 (x, y) からウィンドゥ座標 (nx, ny) への変換は，次の式により行なわれる事を意味します:

[11)] EGGX/ProCALL のビルド時に Xinerama が有効にされていた場合は，最小サイズのディスプレイのピクセルサイズと比較されます.

$\texttt{nx} = \texttt{nxw} + (x - \texttt{xa}) \cdot \texttt{xscale}$

$\texttt{ny} = \texttt{nyw} + (y - \texttt{ya}) \cdot \texttt{yscale}$

ProCALL の描画ルーチンは，アプリケーション座標系 (座標値は実数) で指定するので，このルーチンを一度使えば，座標系の変換は各描画ルーチンが自動的に行なうようになります.

次の使用例では，ウィンドゥの左下をアプリケーション座標系の (-40.0, -20.0) に対応させ，スケーリングファクターを縦・横とも 2.0 に設定します.

使用例　`call newcoordinate(nwin, 0,0, -40.0,-20.0, 2.0,2.0)`

座標系の変更には，newwindow ルーチン (§3.4.6) を使う方法もあります．ご検討ください.

3.4.6 newwindow(nw,xs,ys,xe,ye)

機　能　アプリケーション座標系の変更 (左下と右上の座標を与える)

`nw` で指定したウィンドゥのアプリケーション座標系を変更します (実際のグラフィックスエリアの大きさが変わるわけではありません).

ウィンドゥ上のウィンドゥ座標系 (座標値は整数) は，左下が (0, 0) で，右上が (`nxsize-1`, `nysize-1`) であり，デフォルトではアプリケーション座標系 (座標値は実数) の座標値はウィンドゥ座標系のそれに一致しています.

newwindow ルーチンを使う事により，アプリケーション座標系の左下 (つまりウィンドゥ座標系での (0, 0)) を (`xs`, `ys`)，右上を (`xe`, `ye`) に変更できます.

ProCALL の描画ルーチンは，アプリケーション座標系 (座標値は実数) で指定するので，このルーチンを一度使えば，座標系の変換は各描画ルーチンが自動的に行なうようになります.

次の使用例では，アプリケーション座標系の左下を (-20.0, -10.0)，右上を (799.0, 599.0) に変更します.

使用例　`call newwindow(nwin, -20.0, -10.0, 799.0, 599.0)`

座標系の変更には，newcoordinate ルーチン (§3.4.5) を使う方法もあります．ご検討ください.

3.4.7 layer(nw,lys,lyw)

機　能　レイヤの設定をする

ProCALL ではグラフィックス用ウィンドゥ毎に 8 枚のレイヤを持ち，表示するレイヤと書き込むレイヤを独立に指定できます．`nw` にはウィンドウ番号を指定し，`lys` には表示するレイヤ番号，`lyw` には書き込むレイヤ番号を 0〜7 の整数で指定します.

なお，現在表示しているレイヤに対して (`lys = lyw` の場合に) 連続して描画ルーチンを実行すると，描画パフォーマンスが得られない事があります．高速な描画が必要な場合には，現在表示していないレイヤに対して描画し，copylayer ルーチン (§3.4.8) で描画レイヤの画像を表示レイヤにコピーするようにします.

デフォルトでは `layer(nw,0,0)` の状態となっています.

使用例　`call layer(nwin,0,1)`

3.4.8 copylayer(nw,lysrc,lydest)

機　能　レイヤのコピーをする

`nw` のウィンドウ番号の，レイヤ `lysrc` の画像をレイヤ `lydest` にそのままコピーします．このコピーは瞬時に行われるため，アニメーションの再生に使うことができます.

使用例　`call copylayer(nwin,1,0)`

3.4.9 gsetbgcolor(nw,strc)

機 能 ウィンドゥの背景色を変更する

nw で指定されたウィンドゥの背景色 (gclr ルーチンで初期化される色) を変更します.

strc には X サーバの rgb.txt[12] に設定されている色を指定し,「'Blue'//CHAR(0)」のように必ず最後に「CHAR(0)」を追加します. また,「'#c0c0ff'//CHAR(0)」のように, 16 進数の Red,Green,Blue での指定も可能です.

使用例
```
call gsetbgcolor(nwin,'white'//CHAR(0)) ;
```

3.4.10 gclr(nw)

機 能 描画レイヤの全消去

gsetbgcolor ルーチン (§3.4.9) で指定した色で, 描画レイヤを初期化します. gsetbgcolor での指定がない場合の色は黒となっています.

使用例
```
call gclr(nwin)
```

3.4.11 tclr

機 能 端末のクリア

端末をクリアし, カーソルの位置をホームポジションに戻します.

使用例
```
call tclr
```

3.4.12 newpencolor(nw,nc)

機 能 描画色の変更

nw で指定したウィンドゥでの描画色を変更します. nc と色との関係は以下の通りです.

0:黒 1:白 2:赤 3:緑 4:青 5:シアン 6:マゼンタ 7:黄

8:DimGray 9:Gray 10:red4 11:green4 12:blue4 13:cyan4 14:magenta4 15:yellow4

red4, green4…の "4" のつく色は, 暗い赤, 暗い緑…となっています.

デフォルトでは, 白が指定されています.

使用例
```
call newpencolor(nwin,2)
```

3.4.13 newcolor(nw,strc)

機 能 描画色の変更

nw で指定したウィンドゥでの描画色を変更します. strc には X サーバの rgb.txt[13] に設定されている色を指定し,「'Blue'//CHAR(0)」のように必ず最後に「CHAR(0)」を追加します. また,「'#c0c0ff'//CHAR(0)」のように, 16 進数の Red,Green,Blue での指定も可能です.

使用例
```
call newcolor(nwin,'Violet'//CHAR(0))
```

[12] rgb.txt は UNIX 系の OS なら /usr/X11R6/lib/X11/ などにあります.
[13] rgb.txt は UNIX 系の OS なら /usr/X11R6/lib/X11/ などにあります.

3.4.14 newrgbcolor(nw,nr,ng,nb)

機　能　描画色の変更

nw で指定したウィンドゥでの描画色を変更します．nr,ng,nb にはそれぞれ Red,Green,Blue の輝度を 256 段階の整数 (0〜255) で指定します．

使用例　`call newrgbcolor(nwin,255,127,0)`

3.4.15 newhsvcolor(nw,nh,ns,nv)

機　能　描画色の変更

nw で指定したウィンドゥでの描画色を変更します．nh,ns,nv にはそれぞれ，Hue, Satulation, Value を指定します．ns と nv は 256 段階の整数 (0〜255) を，nh は 0〜359 までの整数 (角度) を指定します．

使用例　`call newhsvcolor(nwin,120,250,240)`

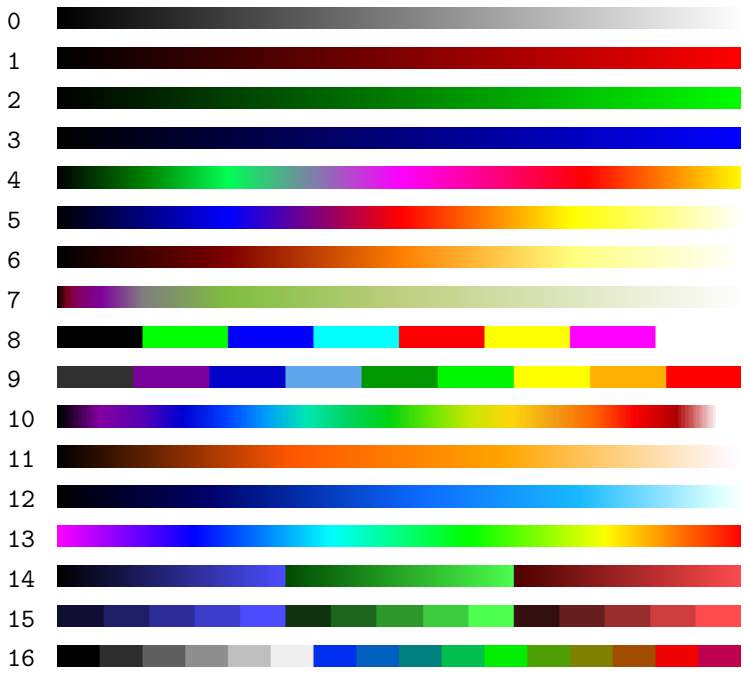
3.4.16 makecolor(ncolormode,dmin,dmax,data,nr,ng,nb)

機　能　変数からカラーを生成する (カラーバー生成)

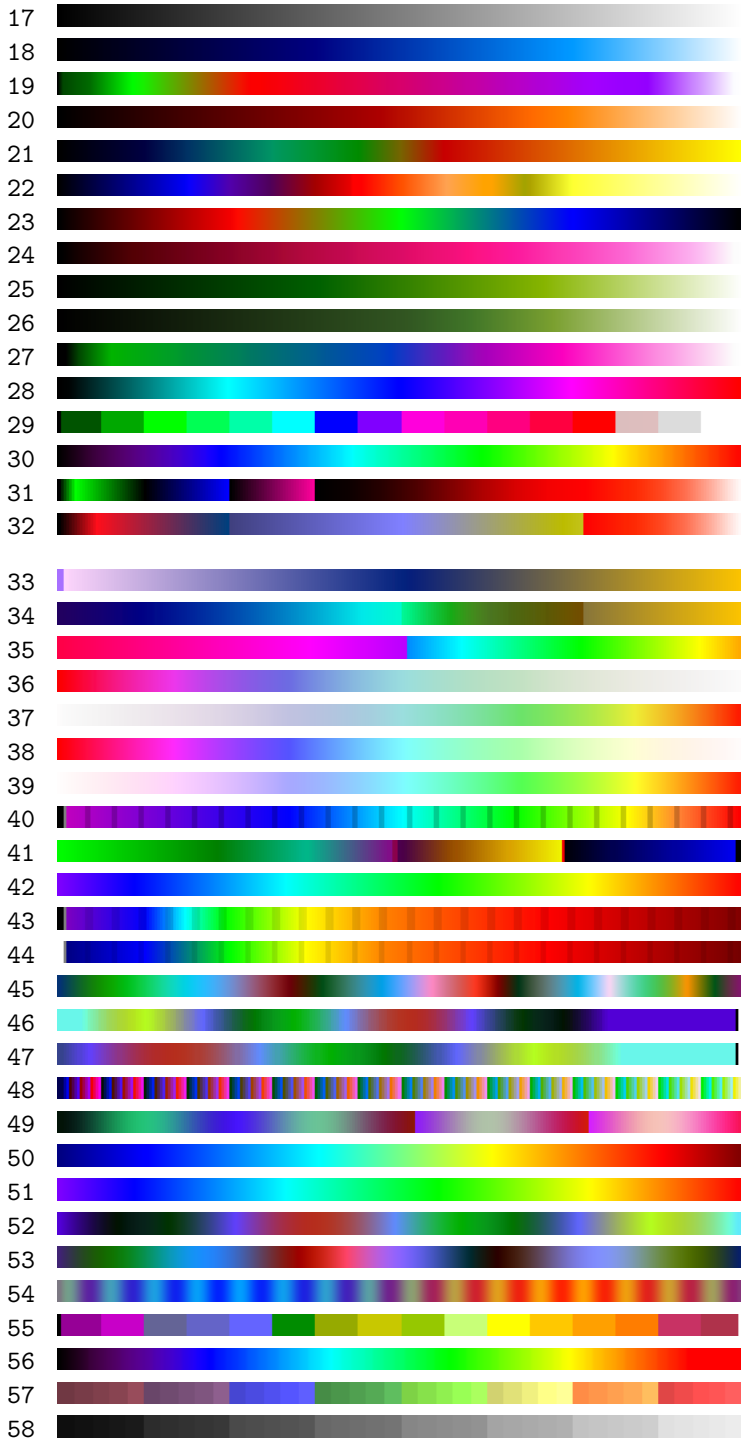
変数 data の値を使って，nr,ng,nb それぞれに 256 段階の Red,Greed,Blue のカラーを生成します．変数の最小，最大は dmin,dmax で指定します．このルーチンで得た，nr,ng,nb の値は newrgbcolor ルーチン (§3.4.14) にそのまま与えて使う事ができます．

ncolormode はカラーパターンの番号で，以下に示す約 50 種類のカラーパターンが利用できます．

以下は fits ビューア DS9 コンパチのカラーパターンです．



以下は可視化ソフトウェア IDL コンパチのカラーパターンです．



使用例　`call makecolor(10,v_min,v_max,v,nr,ng,nb)`

3.4.17　newlinewidth(nw,nwidth)

機　能　線幅の変更

`nw` で指定したウィンドゥで線を描く時の線幅を変更します．デフォルトでは幅 1 が設定されています．

このルーチンで線幅を変更すると，drawsym ルーチン (§3.4.33) や drawarrow ルーチン (§3.4.35) 等で描かれる図形にも影響を与えるので注意してください．

使用例　`call newlinewidth(nwin, 2)`

3.4.18　newlinestyle(nw,nstyle)

機　能　線のスタイルの変更

`nw` で指定したウィンドゥで線を描く時のスタイルを変更します．引数 `style` に与える事ができる値は，`0`(実線) と `1`(点線) です．デフォルトでは実線が設定されています．

このルーチンで線のスタイルを変更すると，drawsym ルーチン (§3.4.33) や drawarrow ルーチン (§3.4.35) 等で描かれる図形にも影響を与えるので注意してください．

使用例　`call newlinestyle(nwin, 1)`

3.4.19　pset(nw,xg,yg)

機　能　点の描画

`nw` で指定したウィンドゥに点を描画します．

描画ルーチン等で用いられるアプリケーション座標系は，デフォルトでは左下が (`0.0`, `0.0`) で右上が (`nxsize-1.0`, `nysize-1.0`) になっています．この座標は，newcoordinate ルーチン (§3.4.5) または newwindow ルーチン (§3.4.6) で変更が可能です．

使用例　`call pset(nwin,x,y)`

3.4.20　drawline(nw,xg0,yg0,xg1,yg1)

機　能　直線の描画

`nw` で指定したウィンドゥの (`xg0`, `yg0`) から (`xg1`, `yg1`) に直線を描画します．

使用例　`call drawline(nwin,x0,y0,x1,y1)`

3.4.21　moveto(nw,xg,yg), lineto(nw,xg,yg)

機　能　連続的に直線を描く

lineto ルーチンを複数回使う事により，`nw` で指定したウィンドゥに連続的に直線を描画します．

moveto ルーチンは，(`xg`, `yg`) を lineto ルーチンのための初期位置に設定します．lineto ルーチンは，以前 `moveto` または `lineto` が呼ばれた時に指定された座標から，(`xg`, `yg`) へ直線を引きます．`moveto` でペンを上げて移動，`lineto` でペンを下ろして描画，と考えるとわかりやすいでしょう．

`xg`，`yg` は実数型の引数です．

使用例　`call lineto(nwin,x,y)`

3.4.22　line(nw,xg,yg,mode)

機　能　連続的に直線を描く

このルーチンは，§3.4.21 の `moveto`，`lineto` ルーチンと同じ動作をします．

新しいプログラムでは，moveto・lineto ルーチンを使用してください．

line ルーチンを複数回使う事により，nw で指定したウィンドゥに連続的に直線を描画します．mode に 2 を指定すると以前 line ルーチンが呼ばれた時に指定された座標から，(xg, yg) へ直線を引きます．mode に 3 を指定すると (xg, yg) を line ルーチンの初期位置に設定します．mode=2 でペンを下ろして描画，mode=3 でペンを上げて移動と考えるとわかりやすいでしょう．

xg，yg は実数型の引数です．

使用例　`call line(nwin,x,y,2)`

3.4.23　drawpts(nw,x,y,n)

機　能　複数の点を描く

nw で指定したウィンドゥで，n 個の点を描きます．x，y は n 個の実数の一次元配列で，x(1)〜x(n)，y(1)〜y(n) に各点の座標を入れておきます．

使用例　`call drawpts(nwin,x,y,5)`

3.4.24　drawlines(nw,x,y,n)

機　能　折れ線を描く

nw で指定したウィンドゥで，折れ線を描きます．x，y は n 個の実数の一次元配列で，x(1)〜x(n)，y(1)〜y(n) に折れ線の各点の座標を入れておきます．

使用例　`call drawlines(nwin,x,y,5)`

3.4.25　drawpoly(nw,x,y,n)

機　能　多角形を描く

nw で指定したウィンドゥで，多角形を描きます．x，y は n 個の実数の一次元配列で，x(1)〜x(n)，y(1)〜y(n) に多角形の各点の座標を入れておきます．

使用例　`call drawpoly(nwin,x,y,5)`

3.4.26　fillpoly(nw,x,y,n,i)

機　能　多角形を塗り潰す

nw で指定したウィンドゥで，多角形の領域を塗り潰します．x，y は n 個の実数の一次元配列で，x(1)〜x(n)，y(1)〜y(n) に多角形の各点の座標を入れておきます．i は塗り潰す時の形状で通常は 0 を，凸多角形の時は 1 を指定します．

使用例　`call fillpoly(nwin,x,y,5,0)`

3.4.27　drawrect(nw,x,y,w,h)

機　能　長方形を描く

nw で指定したウィンドゥに，頂点 (x, y) から正の方向に幅 w，高さ h の長方形を描きます．

使用例　`call drawrect(nwin,50.0,60.0,30.0,20.0)`

3.4.28 fillrect(nw,x,y,w,h)

機　能　長方形の領域を塗り潰す
nw で指定したウィンドゥで，頂点 (x, y) から正の方向に幅 w，高さ h の長方形の領域を塗り潰します.

使用例　call fillrect(nwin,50.0,60.0,30.0,20.0)

3.4.29 drawcirc(nw,xcen,ycen,xrad,yrad)

機　能　中心座標と半径を与えて円を描く
nw で指定したウィンドゥに，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円を描きます.

使用例　call drawcirc(nwin,50.0,60.0,30.0,40.0)

3.4.30 fillcirc(nw,xcen,ycen,xrad,yrad)

機　能　中心座標と半径を与えて円を塗り潰す
nw で指定したウィンドゥに，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円を塗り潰します.

使用例　call fillcirc(nwin,50.0,60.0,30.0,40.0)

3.4.31 drawarc(nw,xcen,ycen,xrad,yrad,sang,eang,idir)

機　能　円の中心，半径，始点，終点の角度を与えて円弧を描く
nw で指定したウィンドゥに，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円弧を描きます．sang は開始角，eang は終了角で，度で与えます．idir は円弧を描く方向で 1 で左廻り，-1 で右廻りとなります.

使用例　call drawarc(nwin,50.0,60.0,30.0,40.0,-10.0,-170.0,-1)

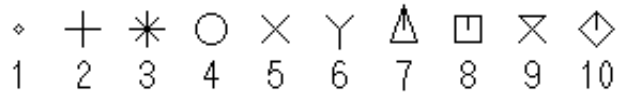
3.4.32 fillarc(nw,xcen,ycen,xrad,yrad,sang,eang,idir)

機　能　円の中心，半径，始点，終点の角度を与えて円弧を塗り潰す
nw で指定したウィンドゥで，(xcen, ycen) を中心に横方向の半径 xrad，縦方向の半径 yrad の円弧を塗り潰します．sang は開始角，eang は終了角で，度で与えます．idir は円弧を描く方向で 1 で左廻り，-1 で右廻りとなります.

使用例　call fillarc(nwin,50.0,60.0,30.0,40.0,-10.0,-170.0,-1)

3.4.33 drawsym(nw,xg,yg,size,nsym)

機　能　センターシンボルの描画
nw で指定したウィンドゥにセンターシンボルを座標 (xg, yg) に描きます．size はシンボルの大きさでピクセル単位の実数，nsym でシンボルの種類 (整数) を指定します.
nsym とシンボルの関係は，次の図のとおりです.



なお，xg，yg は実数型の引数です.

使用例　`call drawsym(nwin,x,y,16.0,2)`

3.4.34　drawsyms(nw,x,y,n,size,nsym)

機　能　複数のシンボルを描く

nw で指定したウィンドゥで，n 個のシンボルを描きます．x，y は n 個の実数の一次元配列で，x(1)〜x(n)，y(1)〜y(n) に各シンボルの座標を入れておきます.

size はシンボルの大きさでピクセル単位の実数，nsym でシンボルの種類 (整数) を指定します.

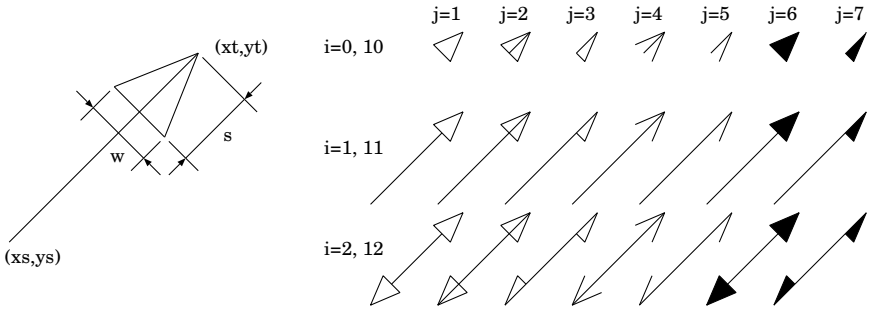
nsym とシンボルの関係については，§3.4.33をご覧ください.

使用例　`call drawsyms(nwin,x,y,5,16.0,8)`

3.4.35　drawarrow(nw,xs,ys,xt,yt,s,w,10*i+j)

機　能　種々の型の矢印を描く

nw で指定したウィンドゥに (xs, ys) から (xe, ye) に向かって矢印を描きます．矢印の形状は以下の図の通りで，s と w は実数で指定します．i が 0〜2 の場合には w,s はピクセル数を，i が 10〜12 の場合には w,s は矢印の長さに対する割合で 0.0〜1.0 の値を指定します.



使用例　`call drawarrow(nwin,x0,y0,x1,y1,0.3,0.2,114)`

3.4.36　newfontset(nw,fontset,nstatus)

機　能　フォントセット (日本語フォント) の指定

nw で指定したウィンドゥで描くフォントセットを指定します．文字の描画は drawstr ルーチン (§3.4.37) を利用します.

フォントセット名は fontset で指定します．fontset の最終文字は「CHAR(0)」(終端文字) でなければなりません.

最後の引数 status には，フォントセットの取得状況を示す整数値が代入されます．fontset で指定したフォントセットが取得できた場合は 0，代替フォントで取得できた場合は正の値，フォントセットの取得に失敗した場合は負の値が status に返ってきます.

フォントセットの設定は，X サーバにインストールされたのフォントを指定する必要があり，OS やディストリビューションに依存します．確実に表示したい場合は，次のような設定を推奨します:

14 ドットフォント	'-*-fixed-medium-r-normal--14-*'//CHAR(0)
16 ドットフォント	'-*-fixed-medium-r-normal--16-*'//CHAR(0)
24 ドットフォント	'-*-fixed-medium-r-normal--24-*'//CHAR(0)

使用例　`call newfontset(nwin,'-*-fixed-medium-r-normal--16-*'//CHAR(0),nstat)`

3.4.37 drawstr(nw,xg,yg,size,str,theta,len)

機　能　文字列の描画

nw で指定したウィンドゥに，文字列を座標 (xg, yg) から描きます．size は文字の大きさで，ピクセル単位の実数で指定します．str に文字列を，len には文字列の長さ (整数) を与えます．文字列 str の最終文字が終端文字「CHAR(0)」であれば，len に実際の文字列長のかわりに-1 を与えてもかまいません．theta は文字列の回転を指定する実数の引数ですが，現バージョンでは機能しません．

文字のサイズ size は 1〜24 の範囲で指定できます．size と実際のフォントとの関係は以下の表のようになっています．この場合，文字は半角英数字のみ描画できます．

マルチバイト文字 (漢字) を描画する場合は，size には 0.0 を指定します．この場合のフォントの指定は，newfontset ルーチン (§3.4.36) を利用します．newfontset でのフォント指定がなされていない場合は，デフォルトの 14 ドットのフォントセットで描画されます．

1〜7 : 5×7	8 : 5×8	9 : 6×9	10〜11 : 6×10	12 : 6×12
13 : 7×13	14〜15 : 7×14	16〜19 : 8×16	20〜23 : 10×20	24 : 12×24

使用例　`call drawstr(nwin,x,y,16.0,'Hoge',0.0,4)`

使用例　`call drawstr(nwin,x,y,0.0,' 日本語の描画も OK!'//CHAR(0),0.0,-1)`

3.4.38 drawnum(nw,xg,yg,size,v,theta,n)

機　能　変数の値を描く

nw で指定したウィンドゥに実数型変数 v の値を座標 (xg, yg) から描きます．size は文字列の大きさで，ピクセル単位の実数で指定します．n は表示する小数点の桁数で，整数値を与えます．theta は文字列の回転を指定する実数の引数ですが，現バージョンでは機能しません．

使用例　`call drawnum(nwin,x,y,16.0,prm,0.0,3)`

3.4.39 putimg24(nw,x,y,nw,nh,nbuf)

機　能　整数配列に用意した画像をウィンドウに一括転送する

nw で指定したウィンドゥの座標 (x, y) に nbuf に用意した，幅 nw，高さ nh の画像を一括転送します．

整数配列 nbuf には，Red,Green,Blue の順に 3 つずつ，水平方向に走査しながら画像の上から下へ用意します (nbuf(1)=nRed(1)，nbuf(2)=nGreen(1)，nbuf(3)=nBlue(1)，といった具合いです)．配列には 0〜255 の範囲の輝度を与えておきます．

なお X サーバの depth が 8 以下の場合はこのルーチンは使用できません (正常に動作しません)．X サーバの depth は ggetdisplayinfo ルーチン (§3.4.1) で調べる事ができます．

使用例　`call putimg24(nwin,x,y,640,400,nbuffer)`

3.4.40 saveimg(nw,ly,xs,ys,xe,ye,fname,n,conv,nd)

機 能 特定のレイヤの画像をコンバータ (netpbm 等) を通してファイルに保存する

バックグラウンドプロセスを起動し，ウィンドウ番号 nw，レイヤ番号 ly の (xs, ys) から (xe, ye) の範囲をコンバータ (netpbm の各種コマンド,ImageMagick の convert 等) を通してファイルに保存します[14]．ファイル名は fname で指定し,「'image.png'//CHAR(0)」のように必ず最後に「CHAR(0)」(終端文字) を追加します．n に正の整数，例えば n=12 を指定すると，実際のファイル名は image12.png のように数字がつき，n が負の場合は数字はつかずにそのままのファイル名で保存します．conv には ppm 形式から各画像フォーマットに変換するコンバータのコマンドラインを指定します．例えば，netpbm を利用して png 形式に保存する場合は,「'pnmtopng'//CHAR(0)」とします．ImageMagick を利用する場合は,「'convert'//CHAR(0)」とします．もちろん，オプションスイッチも含める事もでき,「'pnmtops -scale 0.125'//CHAR(0)」といった指定も可能です．最後の引数 nd は減色パラメータで，R,G,B 1 チャンネルあたりの色の段階数を整数で指定します．nd は簡単な図形では 16 くらいで十分ですが，多くの色を使っている場合は最大値の 256 を指定してください．

ppm から他の画像フォーマットに変換する netpbm のコマンドの例を挙げておきます．各コンバータの使用方法に関しては端末から man コンバータ名 とタイプする事で調べることができます．ImageMagick の convert コマンドを利用する場合は，ファイル名のサフィックス (ドット以降の文字列，例えば.jpg，.png，.eps など) から保存フォーマットが決定されます．

画像フォーマット	コンバータ名
AutoCAD DXB	ppmtoacad
windows bitmap	ppmtobmp
Berkeley YUV	ppmtoeyuv
GIF	ppmtogif
NCSA ICR graphics	ppmtoicr
IFF ILBM	ppmtoilbm
Interleaf image	ppmtoleaf
HP LaserJet PCL 5 Color	ppmtolj
map	ppmtomap
Mitsubishi S340-10 printer	ppmtomitsu
Atari Neochrome .neo	ppmtoneo
PPC Paintbrush	ppmtopcx
portable graymap	ppmtopgm
Atari Degas .pi1	ppmtopi1
Macintosh PICT	ppmtopict
HP PaintJet	ppmtopj
HP PaintJet XL PCL	ppmtopjxl
X11 “puzzle”	ppmtopuzz
three portable graymaps	ppmtorgb3
DEC sixel	ppmtosixel
TrueVision Targa	ppmtotga

画像フォーマット	コンバータ名
Motif UIL icon	ppmtouil
Windows .ico	ppmtowinicon
XPM format	ppmtoxpm
Abekas YUV	ppmtoyuv
YUV triplets	ppmtoyuvsplit
DDIF	pnmtoddif
FIASCO compressed	pnmtofiasco
FITS	pnmtofits
JBIG	pnmtojbig
JFIF (”JPEG”) image	pnmtojpeg
Palm pixmap	pnmtopalm
plain (ASCII) anymap	pnmtoplainpnm
Portable Network Graphics	pnmtopng
PostScript	pnmtops
Sun raster	pnmtorast
RLE image	pnmtorle
sgi image	pnmtosgi
Solitaire image recorder	pnmtosir
TIFF file	pnmtotiff
CMYK encoded TIFF	pnmtotiffcmyk
X11 window dump	pnmtoxwd

ここに挙げた以外のコンバータも，標準入力から ppm 形式で入力し，標準出力から変換データを出力するものであれば使用する事ができます．

コンバータを通さずに直接 ppm 形式で保存する場合は conv には「CHAR(0)」と指定してください．ただしその場合，非圧縮のバイナリデータが保存されるので，ファイルのサイズが大きくなります．

このルーチンでは X サーバから画像データを転送し，保存するわけですが，ネットワークが遅かったりすると非常に時間がかかります．その事を考慮してこのルーチンでは子プロセスを起動して，バックグラウンドで X サーバからの画像の転送・ディスクへの書き込みを行うようにしています．したがってプログラムは call saveimg(…) の後すぐに他の作業ができるようになります．ただし，この子プロセスが終わる

[14] netpbm は http://sourceforge.net/projects/netpbm/ ，ImageMagick は http://www.imagemagick.org/ などで配布されています．

まではXサーバでの描画等ができないので，もし call saveimg(…) のすぐ後に ProCALL の描画ルーチンなどを呼んだ場合は画像の転送・保存が完了するまで待たされる事になります.

なお，saveimg を使っている場合は プログラムは必ず gclose(§3.4.3) を call してから終了する ようにしてください.

使用例 `call saveimg(nwin,0, 0.0,0.0, 639.0,399.0, 'img.png'//CHAR(0),i, 'pnmtopng'//CHAR(0),256)`

png 形式に保存する例です．gif 形式の場合は conv を「'ppmtogif'//CHAR(0)」にします.

使用例
```
call saveimg(nwin,0, 0.0,0.0, 639.0,399.0, 'fig.eps'//CHAR(0),-1,
     'pnmtops -noturn -dpi 72 -equalpixels -psfilter -flate -ascii85'//CHAR(0),256)
```

PostScript 形式に保存する例です．netpbm の pnmtops では RunLength 圧縮や GZIP 圧縮 (可逆圧縮) をサポートしています．上記のように「-psfilter -flate -ascii85」をつける事で画質を損うことなく，ファイルサイズを小さくする事ができます.

3.4.41 gsetnonblock(iflag)

機 能 ggetch，ggetevent，ggetxpress ルーチンの動作モードを設定する

デフォルトでは，キーボードやマウスの入力情報を取得するルーチン ggetch，ggetevent，ggetxpress が呼ばれると，ルーチン内部で入力があるまで待ち続けます (ブロッキングモード).

iflag に 1 を設定して gsetnonblock ルーチンが呼ばれるとノンブロッキングモードとなり，ggetch，ggetevent，ggetxpress ルーチンは入力の有無にかかわらずすぐに戻るようになります.

デフォルトのブロッキングモードに戻すには，iflag に 0 を与えてください.

gsetnonblock ルーチンの呼び出しは，ウィンドゥのオープンの前あるいは後のどのタイミングでも有効です.

使用例 `call gsetnonblock(1)`

3.4.42 ggetch(key)

機 能 キーボードから入力された文字を返す

ProCALL で開いたすべてのウィンドゥからのキーボードの入力情報を返します．キーボードから入力された文字のコードが key(整数) に代入されます．ブロッキングモード (デフォルト) ではキー入力があるまで待ち続けますが，ノンブロッキングモードでは入力の有無にかかわらずすぐにこのルーチンから戻ります (動作モードについては §3.4.41の gsetnonblock ルーチンを参照してください)．ノンブロッキングモードにおいて入力が無かった場合は，key に負の値が返ります.

以下が 16 進の文字コード表です．イタリック体の数字が 16 進 2 桁の上位を表します．例えば,「a」は z'61',「A」は z'41' となります.

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0		Home	PageUp	Pause		End	PageDown		BackSpace	Tab				Enter		
1												Esc	→	←	↑	↓
2	Space	!	”	#	$	%	&	’	(	)	∗	+	,	−	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	Delete

z'01'〜z'1A' の中にはいくつか特殊キーが含まれていますが，この区間は「Ctrl+アルファベットキー」のコードと共有となります．例えば,「BackSpace」と「Ctrl+H」のコードはどちらも z'08' となります．z'01'〜z'1A' で空白になっている個所は,「Ctrl+アルファベットキー」にのみコードが割り当てられています．

使用例　`call ggetch(nwin,key)`

3.4.43　ggetevent(nw,ntype,nbutton,xg,yg)

機　能　マウスやキーボードからの入力の情報を返す

ProCALL で開いたすべてのウィンドゥからのマウスやキーボードの入力情報を返します．ブロッキングモード (デフォルト) では入力があるまで待ち続けますが，ノンブロッキングモードでは入力の有無にかかわらずすぐにこのルーチンから戻ります (動作モードについては §3.4.41の gsetnonblock ルーチンを参照してください)．ノンブロッキングモードにおいて入力が無かった場合は，負の値が nw に返ります．

nw には，入力のあったウィンドゥ番号が返るので，ユーザ・プログラム側で意図したウィンドゥからの入力かどうかをチェックできます．

ntype には，マウスのモーションの場合は 6，マウスのボタンが押された場合は 4，マウスのボタンが離された場合は 5，キーボードからの入力の場合は 2 が返ります．

マウスの入力の場合，nbutton にはクリック (on･off) された，または押されているボタンの番号 (1,2,3,…) が代入され，ウィンドゥ上でのマウスポインタの座標 (アプリケーション座標系) が xg，yg に代入されます．

キー入力の場合は，キーコードが nbutton に返ります．このキーコードは ggetch ルーチン (§3.4.42) の key と同一です．

使用例　`call ggetevent(nwin,ntype,nb,x,y)`

3.4.44　ggetxpress(nw,ntype,nbutton,xg,yg)

機　能　マウスからのボタンクリック，キーボードからの入力の情報を返す

ProCALL で開いたすべてのウィンドゥからのマウスのボタンクリック，キータイプの入力情報を返します．ブロッキングモード (デフォルト) では入力があるまで待ち続けますが，ノンブロッキングモードでは入力の有無にかかわらずすぐにこのルーチンから戻ります (動作モードについては §3.4.41の gsetnonblock ルーチンを参照してください)．ノンブロッキングモードにおいて入力が無かった場合は，nw に負の値が返ります．

nw には，入力のあったウィンドゥ番号が返るので，ユーザ・プログラム側で意図したウィンドゥからの入力かどうかをチェックできます．

マウスのボタンクリックがあった場合，ntype には 4 が代入され，キーボードからの入力の場合は 2 が代入されます．

マウスのボタンクリックの場合，nbutton にはクリックされたボタンの番号 (1,2,3,…)，クリックされた時のウィンドゥ上でのマウスポインタの座標 (アプリケーション座標系) が xg，yg に代入されます．

キー入力の場合は，キーコードが nbutton に返ります．このキーコードは ggetch ルーチン (§3.4.42) の key と同一です．

使用例　`call ggetxpress(nwin,ntype,nb,x,y)`

3.4.45　selwin(nw)

機　能　描画するウィンドゥを指定する

カルコンプ互換ルーチンでどのウィンドゥにアクセスするかを指定します．nw には gopen(§3.4.2) で得た，ウィンドゥ番号を指定します．なお，カルコンプ互換ルーチンの plots(§3.5.1) で開いたウィンドゥの番号は 0 となります．デフォルトは 0 です．

使用例　`call selwin(nwin)`

3.5　カルコンプ互換ルーチンのリファレンス

Pro-FORTRAN とコンパチブルなルーチン群です．Pro-FORTRAN や他の GKS，カルコンプ互換の FORTRAN から移行する場合以外は，必要としないと思われます．

3.5.1　plots

機　能　グラフィックス用ウィンドウを開く

このルーチンを呼ぶことで，グラフィックス用ウィンドウを開きます．グラフィックスエリア (ウィンドゥサイズ) は 640 × 400 ピクセルです．グラフィックスエリアを任意にとりたい場合や複数のウィンドウを開きたい場合は `call gopen` を使用してください．

なお，plots ルーチンで呼び出した ProCALL でのウィンドゥ番号は 0 となります (ProCALL 標準ルーチンのウィンドウ番号 nw には 0 を指定します)．

使用例　`call plots`

3.5.2　window(xs,ys,xe,ye)

機　能　座標系の変更

ウィンドゥ上のウィンドゥ座標系 (座標値は整数) は，左下が (0, 0) で，右上が (639, 399) であり，デフォルトではアプリケーション座標系 (座標値は実数) の座標値はウィンドゥ座標系のそれに一致しています．

window ルーチンを使う事により，アプリケーション座標系の左下 (つまりウィンドゥ座標系での (0, 0)) を (xs, ys)，右上を (xe, ye) に変更できます．

下の使用例では，アプリケーション座標系の左下を (-2.0, -1.0)，右上を (5.0, 4.0) に変更します．

使用例　`call window(-2.0, -1.0, 5.0, 4.0)`

3.5.3　newpen(nc)

機　能　描画色の変更

plot などでの描画色を変更します．nc と色との関係は以下の通りです．

0:黒　1:白　2:赤　3:緑　4:青　5:シアン　6:マゼンタ　7:黄

8:DimGray　9:Gray　10:red4　11:green4　12:blue4　13:cyan4　14:magenta4　15:yellow4

red4，green4…の “4” のつく色は，暗い赤，暗い緑…となっています．

デフォルトでは，白が指定されています．

使用例　`call newpen(2)`

互換性　8〜15 の色は本家 Pro-FORTRAN では使用できません．

3.5.4 clsc

機　能　端末のクリア
端末をクリアし，カーソルの位置をホームポジションに戻します.

使用例　`call clsc`

3.5.5 clsx

機　能　グラフィックス画面をクリア

使用例　`call clsx`

3.5.6 plot(xg,yg,mode)

機　能　直線，点の描画
`mode` に 2 を指定すると以前 `plot` が呼ばれた点から，(`xg`, `yg`) へ直線を引きます．`mode` に 3 を指定すると (`xg`, `yg`) を plot ルーチンの初期位置に設定します．`mode=2` でペンを下ろして描画，`mode=3` でペンを上げて移動と考えるとわかりやすいでしょう.
また，`mode=1` の場合は (`xg`, `yg`) に点を描き，ペンの位置を更新します.
`xg`，`yg` は実数型の引数です.

使用例　`call plot(x,y,2)`

互換性　`mode=1` は本家 Pro-FORTRAN では使用できません.

3.5.7 arc(xcen,ycen,rad,sang,eang,idir)

機　能　円の中心，半径，始点，終点の角度を与えて円弧を描く
(`xcen`, `ycen`) を中心に半径 `rad` の円弧を描きます．`sang` は開始角，`eang` は終了角で，度で与えます．`idir` は円弧を描く方向で `1` で左廻り，`-1` で右廻りとなります.

使用例　`call arc(50.0,60.0,30.0,-10.0,-170.0,-1)`

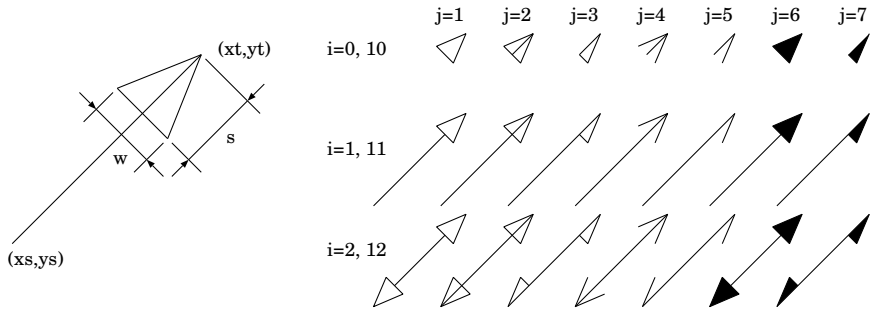
3.5.8 circ1(xc,yc,r)

機　能　中心座標と半径を与えて円を描く
(`xc`, `yc`) を中心に半径 `r` の円を描きます.

使用例　`call circ1(50.0,60.0,30.0)`

3.5.9 arohd(xs,ys,xt,yt,s,w,10*i+j)

機　能　種々の型の矢印を描く
(`xs`, `ys`) から (`xe`, `ye`) に向かって矢印を描きます．矢印の形状は以下の図の通りで，`s` と `w` は実数で指定します．`i` が 0〜2 の場合には `w`,`s` はピクセル数を，`i` が 10〜12 の場合には `w`,`s` は矢印の長さに対する割合で 0.0〜1.0 の値を指定します.



使用例　`call arohd(x0,y0,x1,y1,0.3,0.2,114)`

互換性　`i`=10〜12 は本家 Pro-FORTRAN では使用できません．本家 Pro-FORTRAN では `window` の指定によって実際に描画される矢印の先端の大きさが変化しますが，ProCALL では変化しません．

3.5.10　symbol(xg,yg,size,nstr,theta,len)

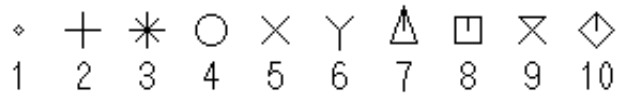
機　能　文字列，センターシンボルの描画

文字列またはセンターシンボルを座標 (xg, yg) に描きます．`size` は文字列・シンボルの大きさで，ピクセル単位の実数で指定します．文字列を描く場合は `len` に文字列の長さ (整数) を，`nstr` に文字列を与えます．センターシンボルを描く場合は `len` に`-1` を指定し，`nstr` にシンボルの番号 1〜10 の整数を与えます．

文字のサイズ `size` は 1〜24 の範囲で指定できます．`size` と実際のフォントとの関係は以下のようになっています．文字は半角英数字ならすべて描画できる上，本家 Pro-FORTRAN より綺麗です．

1〜7 : 5×7	8 : 5×8	9 : 6×9	10〜11 : 6×10	12 : 6×12
13 : 7×13	14〜15 : 7×14	16〜19 : 8×16	20〜23 : 10×20	24 : 12×24

`nstr` とシンボルの関係は，次の図のとおりです．



なお，このルーチンは 4 番目の引数に，文字列型と整数型の両方をとります．この `symbol` を使って文字列，シンボルの両方を描くように作成されたソースでは，コンパイルすると Warning が出てきます．この Warning はかなり目障りなので，本家 Pro-FORTRAN でコンパイルしない場合は，ProCALL 標準ルーチン `drawstr`(文字列を描く), `drawsym`(シンボルを描く) の使用を推奨します．

使用例　`call symbol(x,y,16.0,'Hoge',0.0,4)`
`call symbol(x,y,16.0,2,0.0,-1)`

互換性　本家 Pro-FORTRAN では `window` の指定によって実際に描画されるシンボルの大きさが変化しますが，ProCALL では変化しません．

24 より大きい文字サイズは 12 × 24 ドットフォント固定となります．アルファベットの小文字と一部の記号は本家 Pro-FORTRAN では出力できません．`theta` は文字列の回転を指定する実数の引数ですが，現バージョンでは機能しません．

3.5.11 number(xg,yg,size,v,theta,n)

機　能　変数の値を描く
実数型変数 v の値を座標 (xg, yg) から描きます．size は文字列の大きさで，ピクセル単位の実数で指定します．n は表示する小数点の桁数で，整数値を与えます．

使用例　`call number(x,y,16.0,prm,0.0,3)`

互換性　24 より大きい文字サイズは 12 × 24 ドットフォント固定となります．theta は文字列の回転を指定する実数の引数ですが，現バージョンでは機能しません．

3.5.12 vport,setal

機　能　ダミーのルーチン

互換性　これらのルーチンは機能しません．Pro-FORTRAN のソースをそのままコンパイルできるように，これらのルーチンは形だけ残してあります．

3.6 補助ルーチンのリファレンス

FORTRAN の不便さを少しでも解消するように用意したルーチンです．

3.6.1 msleep(ms)

機　能　ミリ秒単位で実行を延期する
ms ミリ秒の間，プログラムの実行をなにもせずに待ちます．ms には 999 までの整数を指定します．アニメーション速度の調整に利用できます．

使用例　`call msleep(100)`

3.6.2 isnan(v,iflg)

機　能　実数型変数を非数 (Not a Number) かどうか調べる
実数型変数 v を非数かどうかを調べ，非数なら iflg に 0 以外の整数値を返します．

使用例　`call isnan(x,nf)`

3.6.3 rtoc(v,n,ns,str,m)

機　能　実数型変数を文字列に変換する
実数型変数 v を小数点 n 桁まで文字列に変換し，区切り文字'\0' を最後に付け足して文字型変数 str に格納します．ns は str で確保されている文字数を指定します．m には str に文字列を格納した時に余った文字数 (整数値) を返し，この値が負の場合は str で確保されている文字数が足りなかった事を示します．

使用例　`call rtoc(x,4,10,st,m)`