

C++グラフィック・クラスライブラリ: JGR の使い方

平成 17 年 3 月 31 日

目次

1	jgr の特徴	1
2	使用上の注意	2
2.1	ライセンス	2
2.2	その他の注意	2
3	jgr のインストール	2
4	jgr の使いかた	2
5	jgr の仕様	3
5.1	簡単な例	3
6	ウィンドウ及びグラフの初期化の関数	4
6.1	ウィンドウを開く	4
6.2	グラフの初期化	5
6.3	ウィンドウにグラフを登録	6
6.4	ウィンドウの描画	6
6.5	グラフの描画に関する設定	7
6.6	その他初期化に関する関数	7
7	グラフ関連の関数 (描画・マウス・eps)	7
7.1	座標軸関係の描画	8
7.2	グラフ描画関数	9
7.3	マウス入出力	9
7.4	eps 出力関連	10
8	ウィンドウ操作の関数	10
9	すべてのウィンドウに関わる関数	10
10	メニューについて	11
10.1	メニューウィンドウ (Menu Class) について	11
10.2	入力ウィンドウについて	12
10.3	簡単な例	13
10.4	ウィンドウマネージャーに関する注意	14
11	問題点	14
	索引	14

1 jgr の特徴

JGR は X Window 上でグラフィックを描くための C++ クラスライブラリです。複数のウィンドウを管理でき、また、1つのウィンドウ内に複数のグラフを管理できます。プログラムの実行中に随時計算結果をどんどんプロットしていきたい時に便利です。

通常、グラフを描く時には X 軸の座標の単位系 (例えば m,s) における値とグラフィック画面上での値 (ドット) が違いますが、このグラフィック・ツールでは X、Y 軸の長さ、実際の値 (m,s など) と画面上での値 (ドット) の関係などをはじめに与えておくことによって、あとは自動的に実際の値から座標変換を行なって点や線を描いてくれます。Xlib の関数を使っていますが、Xlib を意識せずに使用できます。

2 使用上の注意

2.1 ライセンス

PS 関連ルーチン (jgrPS.cc) は XeasyGraphic-Copyright に、それ以外は LGPL とします。なお、ライブラリに PS 処理を含むかどうかはマクロ変数 PS で制御できます。jgr.h のはじめのほうで定義してます。XeasyGraphic-Copyright に抵触するが、LGPL の範囲で利用する場合はマクロ定数 PS を無効にしてお使いください。

2.2 その他の注意

なにか不具合があるかもしれません。バグレポート・パッチその他大歓迎です。

3 jgr のインストール

インストールは

```
$ make
$ make -C doc
$ make install
```

でどうぞ。

EPS 出力が不要な場合は、jgr.h で

```
#define PS
```

の行をコメントにしてください。若干は軽くなります。(体感できる程度かは不明)

4 jgr の使いかた

- C プログラムで使いたいときにも、コンパイラは g++ を使って下さい。g++ は通常の C プログラムもコンパイルできるので、C のプログラムを書き直す必要はありません。ただし、その時には、

```
#include <stream.h>
```

をインクルードファイルに追加して下さい。

- jgr を使うには、以下のヘッダファイルを include して下さい。

```
#include <jkit/jgr.h>
```

- また、コンパイルするときには libX11, libjgr をリンクして下さい。g++ へのコンパイルオプションは、-lX11 -ljgr になります

具体的なコンパイルの仕方は後の具体例を見て下さい。

5 jgr の仕様

5.1 簡単な例

ここでは簡単な例で説明します。

```
//-----サンプル始まり-----
#include <stream.h>
#include <iostream.h>
#include <jkit/jgr.h>    //jgr クラスを使うのに必要

JWindow win;           //開くウィンドウを宣言
Graph g;               //ウィンドウに描画するグラフを宣言

main(){
double xmax, ymax;
char *msg="Click the Mouse, Please!!";

// ウィンドウと、そこに描くグラフを登録-----
win.open(10,10)  // ウィンドウを (10,10) の位置に開く宣言
    .graph(g.open( 500, 50, 200, 40 ))
        // 開いたウィンドウに、x 軸の正負方向が (500,50)、
        //y 軸の正負方向 (200,40) のグラフを登録
    .map();       // ウィンドウを画面上にマップする。

//ここからは、グラフ g の属性設定-----
xmax = 10.0;
ymax = 100.0;
g.set_ratio( xmax, ymax, 500.0, 200.0 )
    // xmax, ymax の値をそれぞれ、画面上では 500dot, 200dot の長さで描画
    .axis() // X 軸, Y 軸を描画
    .text( 5, -15, msg ) // メッセージをグラフ上の (5,-15) の位置に出力
    .flush()           // グラフィック・バッファにたまってデータをフラッシュ
    .mouse_wait();     // マウス・クリックの入力を待つ

//メインルーチン：グラフ g 上に線を書く.....
for( double i=1.0; i<=xmax; i+=0.1 ){
    g.color("blue") // グラフ g の描画色を青にする
        .line( (double)i, 0.0, 0.0, (xmax-(double)i)*ymax/xmax )
            // (i,0.0) から (0.0, (xmax-(double)i)*ymax/xmax) に線を引く
        .flush();
    mouse_break(); // マウス・クリックがあったら for ループから抜ける
}

g.color("black")
    .text( 5, -15, msg) // テキスト表示
    .mouse_wait();     // マウス・クリックを待つ

// end -----
```

```
Xwin.close();          // 終了の儀式
}
//-----サンプル終了-----
```

このプログラム例は、sample ディレクトリに tiny.cc という名前で置いてあります。このプログラムのコンパイルは以下のコマンドで行います。

```
g++ -o tiny -lm -L /usr/X11R6/lib -lX11 -ljgr tiny.cc
```

またサンプルプログラムには Makefile もあるので、

```
make tiny
```

とすると tiny.cc がコンパイルされ、

```
make
```

とすると、サンプルプログラム全部がコンパイルされます。

サンプルプログラムのうち、sample.cc は tiny.cc にさらに円や四角などの描画のルーチンを増やしたもので、sin.cc は、複数の窓を開いて、また一つの窓に複数のグラフを書いたりするサンプルです。sin.cc では描画したグラフの eps 出力も行います。なんとなく tiny.cc がわかったらこれらのプログラムの中身も見てみてください。以降の節では、プログラムに用いられる各関数について紹介します。

6 ウィンドウ及びグラフの初期化の関数

この節では、ウィンドウやそこに描画するグラフを初期化する関数について、ほぼプログラム内で宣言する順番で紹介して行きます。

6.1 ウィンドウを開く

```
JWindow& open( unsigned int sx, unsigned int sy,
               unsigned int width, unsigned int height, int bsflag=True );
JWindow& open( unsigned int sx, unsigned int sy, int bsflag=True );
```

- (sx,sy): 左上の角の座標、原点は画面の左上の角
- (width,height): ウィンドウの幅と高さ
- bsflag: バッキングストアを使うかどうか (指定がなければ” 使う”)

ウィンドウのクラス名は JWindow です。例えば2つウィンドウを開きたいときには、

```
JWindow win1, win2;
```

と宣言します。さらに、このウィンドウの大きさや位置を以下のように指定します。

- 大きさと位置両方を指定するとき、

```
win1.open(10,10,200,100); //座標 (10,10) を左上の角とし、幅 200dot, 高さ 100dot
```

- 位置のみ指定し、大きさは登録するグラフの大きさに応じて自動的に決める時。

```
win1.open(10,10); //座標 (10,10) を左上の角とする
```

また、バッキングストアの指定を付け加えたいときには、True か False を指定した引数を加えてください。この指定は独立に

```
JWindow& depend_wm(int wmflag=False);
```

という関数を用いて行うこともできます。

6.2 グラフの初期化

```
Graph& open(int sx, int sy, int x_pos, int x_neg, int y_pos, int y_neg,
            char* clr="black" );
Graph& open( int x_pos, int x_neg, int y_pos, int y_neg,
            char* clr="black" );
Graph& open( Graph& g, char* clr="black" );
Graph& child( Graph& g );
Graph& child( Graph& g, char* clr );
```

- (sx,sy): グラフの左上の角の座標、原点はウィンドウの左上の角
- x_pos,x_neg: X 軸の正方向・負方向の長さ
- y_pos,y_neg: Y 軸の正方向・負方向の長さ
- clr: グラフの描画色 ("red","blue" など X window で使える色名を指定)

グラフのクラス名は Graph です。それぞれのグラフについて宣言を以下のように行います。例えば2つグラフを書きたいときには(1つのウィンドウ上であっても、3つのウィンドウそれぞれに書くのであっても)、

```
Graph grp1, grp2 ,grp3;
```

と宣言します。次に、これらのグラフの初期化を行います。

- ウィンドウ中でのグラフの位置、大きさを指定するとき
例えば、ウィンドウ中、(10,10)の位置に X 軸の正負の方向の長さがそれぞれ 100,10, Y 軸の正負の方向の長さがそれぞれ 50,10 のグラフを書きたいときには

```
grp1.open(10,10,100,10,50,10);
```

とします

- ウィンドウ中でのグラフの大きさだけ指定して、位置は自動で決める場合には以下のようにします。

```
grp2.open(100,10,50,10); //大きさは grp1 と同じ場合の例
```

- あるグラフと同じ大きさのグラフを書きたい時には、

```
grp3.open(grp2);
```

とします。この例では grp3 の大きさは grp2 と同じになり、位置は自動で決められます。

いずれの場合にも、描画色の指定をしたいときには、さらに"red", "blue"などを指定した引数を加えてください。この指定は

```
Graph& color( char* clr="black" );
```

を使って、いつでも変更できます。

さらに、同じ位置に重ねてグラフを書きたいときには、そのグラフの初期化に `child` を使います。(1つのグラフに複数の曲線を書く時などに使います) 例えば、グラフ `grp3` を グラフ `grp2` と同じ位置に描く時、

```
grp3.child(grp2);
```

とします。この時、`grp3` は `grp2` の `child graph` と呼びます。`child graph` の描画色は、それぞれ異なる色が自動的に選ばれます。自分で色の設定をしたいときには、

```
grp3.child(grp2, "blue");
```

と、色の指定を加えます。

6.3 ウィンドウにグラフを登録

```
JWindow& graph( Graph& g );
```

前節で初期化したグラフを、描画するウィンドウに登録します。例えば、グラフ `grp1` をウィンドウ `win1` に描きたい場合には、

```
win1.graph(grp1);
```

とします。また、`grp1` の初期化とあわせて、

```
win1.graph(grp1.open(100,10,50,10));
```

などとすることもできます。

6.4 ウィンドウの描画

```
JWindow& map( void );
```

ウィンドウにグラフの登録がすんだところで、ウィンドウをいよいよ X ウィンドウ上に描画します。単に

```
win1.map();
```

とするだけです。ただし、これだけだと、X window の処理の問題上、データがグラフィックバッファに入るだけですぐに描画されない時があります。そのときには、以下の命令を `map` コマンドのあとに入れてください。

```
win1.flush();
```

これで、バッファにたまっているデータがすべて描写されます。なおこの命令は、`jgr` のデフォルトで定義される画面制御のためのクラス `Xwin` の関数 (see sec.9) により、

```
Xwin.flush();
```

を用いるのと等価です。

6.5 グラフの描画に関する設定

```
Graph& set_ratio( double realx, double realy,
                  double scrx, double scry );
Graph& set_ratio( Graph& g );
Graph& set_ratio( void );           // for child graph
```

- (realx, realy): 実際の数値
- (scrx, scry): スクリーン上での値 (ドット)

計算で得られる数値の大きさと、画面上での大きさの関係を設定します。例えば、グラフ grp1 上で、X 軸の時間 10s を画面上で 100dot、Y 軸の速度 5m/s を画面上で 60dot で表したい時には、

```
grp1.set_ratio(10,5,100,60);
```

とします。また grp2 での設定は grp1 と同じにしたい時には、

```
grp1.set_ratio(grp2);
```

さらに、グラフを重ねて描画している child グラフの場合には単に

```
grp3.set_ratio();
```

とすれば、親グラフと同じ設定になります。

この関数の設定後、線などを描画する時の座標の入力は、スクリーン上での大きさを気にすることなく、ただ、(時間、速度) といった実際の値を入力すれば描画ができます。

なお、この関数を宣言しない時の、スクリーン上の値と実際の値の比は 1:1 に設定されています。

6.6 その他初期化に関する関数

```
JWindow& addLeftMargin(long xmargin);
```

ウィンドウに位置の自動割当てでグラフを書いたとき、グラフの左にマージンがもうすこし欲しい場合に使う クラス JWindow の関数です。ウィンドウの map 前に指定してください。

7 グラフ関連の関数 (描画・マウス・eps)

この節では Graph クラスに登録されている関数の説明をします。

7.1 座標軸関係の描画

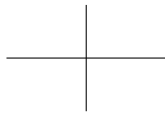
```

Graph& set_axis_with_digit( int flag=True ); //数値付き目盛のある座標軸
Graph& set_xscale_full( int flag=True ); // x 座標軸の目盛刻みは高さ一杯に
Graph& set_yscale_full( int flag=True ); // y 軸の目盛刻みは横幅一杯に
Graph& set_scale_full( int flag=True ); // 座標軸の目盛刻みはグラフ一杯に (両方)
Graph& axis( void ); // 座標軸を書く
Graph& axis_box( void ); // 枠で囲んだ座標軸を書く
Graph& xaxis( double blank ); // 間隔 blank の刻みを座標軸に書く
Graph& yaxis( double blank ); // (指定がなければ自動で間隔は決まる)
Graph& xaxis( void ); // x 軸を書く
Graph& yaxis( void ); // y 軸を書く

```

もし、グラフ grp に関して

```
grp.axis();
```



とすると、という感じの座標軸が書かれます。また、

```
grp.axis_box();
```



とすると、となります。

X 軸だけ、もしくは Y 軸だけ書きたいときには、

```

grp.xaxis( 1.0 ); //目盛の刻み幅を 1.0 にして、
grp.yaxis( 0.1 ); //目盛の刻み幅を 0.1 にして、Y 軸を書く
grp.xaxis(); // X 軸を書く
grp.yaxis(); // Y 軸を書く

```

などします。目盛の刻み幅の指定をしていないときには、目盛は自動設定になります。目盛をグラフ一杯の長さにしたい時には、`set_scale_full()` や `set_xscale_full()`、`set_yscale_full()` を使って、

```

grp.set_xscale_full()
.axis();

```

などします。

目盛に数値を表示したいときには、

```

grp.set_axis_with_digit()
.axis();

```

と、`set_axis_with_digit()` を使います。

7.2 グラフ描画関数

```

Graph& color( char* clr="black" ); //色指定
Graph& pset(double x, double y);   //(x,y) に点をうつ
Graph& set_solid_line( void );     //直線の属性をソリッドに
Graph& set_dash_line( void );     //直線の属性を点線に
Graph& set_double_dash_line( void ); //直線の属性を2点鎖線に
Graph& line(double x, double y, double x2, double y2);
                                   //(x,y) から (x2,y2) に直線
Graph& dash_line( double x, double y, double x2, double y2 );
                                   //(x,y) から (x2,y2) に点線
Graph& line(double x2, double y2); // ラストポイントから (x2,y2) に
Graph& dash_line( double x2, double y2 );
Graph& rectangle( double x, double y, double width, double height );
                                   //左上が (x,y) になる位置に (width,height) の四角を
Graph& fill_rectangle( double x, double y,
                       double width, double height );
                                   //左上が (x,y) になる位置に (width,height) の塗りつぶした四角を
Graph& circle( double x, double y, double r );
Graph& fill_circle( double x, double y, double r );
Graph& forgetLastPoint(void);
                                   //ラストポイントの消去
Graph& forget(void); // 上と同じ
Graph& text( double x, double y, char *msg, int position=Center );
                                   //msg を位置 (x,y) に書く。(x,y) は文字の中心 (デフォルト)
                                   //位置指定に関しては、Left, Center, Right, LeftTop,
                                   //CenterTop, RightTop, RightCenter から選べる。
Graph& clear( void );
                                   //グラフのあるウィンドウを消去
                                   //同じウィンドウ上の全てのグラフのラストポイントも消去します
Graph& flush( void )
                                   //バッファをフラッシュ

```

7.3 マウス入出力

```

int      mouse_press( void );
                                   //グラフのあるウィンドウへのマウス入力があると True を返す。
Graph& mouse_wait();
                                   //グラフのあるウィンドウへのマウス入力を待つ

```

7.4 eps 出力関連

```
void resetPS();
    //PS 出力のための情報を消去
void EPSOut(char* epsfname,
    int lmargin=LEFT_MARGIN, int bmargin=BOTTOM_MARGIN);
    //グラフの EPS をファイル epsfname に書き込む
void EPSAppend(ofstream& epsf,
    int lmargin=LEFT_MARGIN, int bmargin=BOTTOM_MARGIN)
    //グラフの EPS をファイル epsfname に書き足す
```

8 ウィンドウ操作の関数

この節では JWindow クラスに登録されている関数の説明をします。

```
JWindow& map( void );    // ウィンドウをマップ
JWindow& unmap( void ); // ウィンドウをアンマップ
JWindow& clear( void );
    // ウィンドウ内を消去し、ウィンドウ上の全てのグラフのラストポイントを消去
JWindow& axis( void );  // ウィンドウ上の全てのグラフの座標軸を描く
JWindow& flush( void ); // バッファをフラッシュ
int      mouse_press( void ); // ウィンドウをマウス・クリックすると True がかかる
JWindow& mouse_wait();    // ウィンドウへのマウスクリックを待つ
JWindow& expose();        // ウィンドウが expose されたら True を返す
void      EPSOut(char *epsfname); // ウィンドウ上の全てのグラフを eps 出力
Window    get_window( void ); //ウィンドウの情報 (Window) を得る
int        get_index( void ); //ウィンドウの情報 (インデックス) を得る
```

9 すべてのウィンドウに関わる関数

すべてのウィンドウの操作のために、Xwin というクラスがデフォルトで定義されています。(クラス名は XController) これを使うことで、全てのウィンドウに対するいくつかの操作を行えます

```
void      close( void ); // 終了 (最後にウィンドウ全てを閉じる)
XController& flush( void ); //ウィンドウをフラッシュ
XController& sync( void )  //ウィンドウをシンク
int        mouse_press( void ); //どれかのウィンドウへマウスクリックが
    //あれば True を返す
XController& mouse_wait( void )//どれかのウィンドウへマウスクリックが
    //あるのを待つ
XController& expose( void )//どれかのウィンドウが expose されたら True を返す
```

これらは、例えば、

```
Xwin.mouse_press();
```

などとして使います。

10 メニューについて

メニュー関連は上原さん(現東レ所属)の作成をしたものをもとに、手を加えたものです。感謝です。(注: 以下の文書はとても古いもので、現状と一致しません。。現在準備中です。)

10.1 メニューウィンドウ (Menu Class) について

class Menu のオブジェクトは、メニューの要素(アイテム)としてユーザー定義の処理関数とタイトルの組を受けとり、メニューウィンドウにタイトルを表示し、アイテムのタイトル部分がクリックされると該当する処理関数を呼びます。

メニューウィンドウはウィンドウマネージャーの支配を受けます。

Menu は、そのオブジェクトを宣言して直接メッセージを送ることにより使用します。

Menu 自身はその内部にイベント待ちループを持っていません。

タイトルは最大 32 文字で日本語は扱いません。処理関数としては void (*func)(Menu&) 型の関数をとります。アイテムの数に制限はありません。アイテムの数を登録する必要もありません。

ユーザーに公開するメソッドは以下の通りです。

- Menu& open(int x=0,int y=0);
 働き : メニューウィンドウを開く。最初に宣言。
 引数 : メニューウィンドウの ルートでの位置 x,y
 戻り値: 当該オブジェクトの参照
- Menu& addItem(const char* title, void (*handler)(Menu&));
 働き : メニューにアイテムを登録する。
 引数 : アイテムのタイトルの文字列へのポインタ title,
 処理関数へのポインタ handler
 戻り値: 当該オブジェクトの参照
- Menu& addSubmenu(const char* title, Menu& submenu, int type=Temporal));
 働き : メニューにサブメニューを登録する。
 引数 : アイテムのタイトルの文字列へのポインタ title,
 サブメニューへのポインタ submenu,
 サブメニューのタイプ type (Temporal もしくは Static)
 戻り値: 当該オブジェクトの参照
- Menu& addClose(void);
 働き : メニューをクローズするためのメニュー項目を追加する。
 引数 : なし
 戻り値: 当該オブジェクトの参照
- Menu& addExit(void);
 働き : プログラムを終了するためのメニュー項目を追加する。
 引数 : なし
 戻り値: 当該オブジェクトの参照
- Menu& setTitle(const char* str);
 働き : メニュー自体のタイトルを登録する。登録しなければタイトルは 'Menu' となる。
 引数 : タイトルの文字列へのポインタ str,

戻り値：当該オブジェクトの参照

• `Menu& setType(int type=Static);`

働き：メニューのタイプを決める

引数：メニュー項目を選んでもメニューを閉じないとき `Static`、
メニュー項目のクリックがあったときメニューを閉じるなら
`Temporal`

戻り値：当該オブジェクトの参照

• `Menu& setStyle(int style=Separate)`

働き：メニューの概観を決める

引数：メニュー項目の間にしきりをいれるなら `Separate`、いれな
いなら `Flat` を指定する。

戻り値：当該オブジェクトの参照

• `Menu& map(); map(int x,int y);`

働き：メニューウィンドウをマップする。
`open()` 以降で使用する。

引数：メニューウィンドウの ルートでの位置 `x,y`
あるいは、なし
(以前にマップした位置にマップする)

戻り値：当該オブジェクトの参照

• `Menu& unmap();`

働き：メニューウィンドウをアンマップする。
`open()` 以降で使用する。

引数：なし

戻り値：当該オブジェクトの参照

• `Menu& act();`

働き：マウスクリックをチェックし、クリックがあれば処理関数を呼ぶ。
メインループ中で使用。

引数：なし

戻り値：当該オブジェクトの参照

• `Menu& close();`

働き：メニューウィンドウを削除する。とくに明示的に呼ぶ必要はない。
`open()` を呼べば再びメニューウィンドウは使える。

引数：なし

戻り値：当該オブジェクトの参照

使い方の例は次節を参照して下さい。

10.2 入力ウィンドウについて

以下の関数を呼ぶことによって、入力ウィンドウをメニュー項目として登録し、ユーザーから文字列あるいは数値を取得することができます。

• `Menu& addGetString(char* value, char* title, char* dflt);`

働き : 文字変数 `value` に文字を入力する窓をメニュー項目としてつくる。

引数 : 文字変数 `value`, `value` のデフォルト値 `dflt`, メッセージ `title`

戻り値: 当該オブジェクトの参照

• `Menu& addGetValue(double& value, char* title, double dflt);`

働き : 変数 `value` に数値を入力する窓をメニュー項目としてつくる。

引数 : 変数 `value`, `value` のデフォルト値 `dflt`, メッセージ `title`

戻り値: 当該オブジェクトの参照

入力ウィンドウは、文字キー以外に、RET、BS、DEL、ESC、を受け付けます。ESC で入力ウィンドウをクリアします。マウスのボタンプレスは RET と同じ働きをします。

各文字列の最大長は `Menu` の `MAX_STR_LEN` と同じです。

10.3 簡単な例

以下に使い方の例を示します。

```
//
// メニューウィンドウ、入力ウィンドウの使い方の例
//
//   3つのアイテムを持つメインメニューを開く。
//   1つめのアイテムで入力を受け付けるための一時的なサブメニューを開く
//   サブメニューの1つめのアイテムで、文字列を受け付ける。
//   サブメニューの2つめのアイテムで、数値を受け付ける。
//   2つめのアイテムで変数を標準出力に表示するための静的なサブメニューを開く
//   サブメニューの1つめのアイテムで、文字列を標準出力に出す。
//   サブメニューの2つめのアイテムで、数値を標準出力に出す。
//   3つめのアイテムで終了する。
//
//
#include <iostream.h>
#include <stdlib.h>
#include <jkit/jgr.h>

Menu main_menu(Horizontal),
      sub_menuA(Vertical,Static),
      sub_menuB(Vertical,Static);

char str[MAX_STR_LEN+1];
//char str[256];
double v;

////////// サブメニューのためのハンドラ //////////

void handler_subB1(void* dummy){           //文字列を表示する
    cout << "str= " << str << "\n";
}

void handler_subB2(void* dummy){           //数値を表示する
```

```

    cout << "v= " << v << "\n";
}

////////// メインルーチン //////////

int main(){
    sub_menuA.open()
        .addGetString(str,"Input string","aaaa")
        .addGetValue(v,"Input value",10.0)
        .addClose();

    sub_menuB.open()
        .addItem("string",handler_subB1)
        .addItem("value", handler_subB2)
        .addClose();

    main_menu.open(100,50)
        .addSubmenu("input",sub_menuA)
        .addSubmenu("print",sub_menuB)
        .addExit()
        .map();

    while(1) {main_menu.act();}                // メインメニューを走らせる
}

```

このプログラム例は、sample ディレクトリに menu.cc という名前で置いてあります。

10.4 ウィンドウマネージャーに関する注意

メニューウィンドウと入力ウィンドウはウィンドウマネージャーの支配を受けます。これらが開かれる際、アプリケーションが指定するウィンドウ初期位置をウィンドウマネージャーに使用させるために、ウィンドウマネージャーの startup file で、そのことを指定しておかなければなりません。例えば、.twmrc で

```
UsePPosition "on"
```

という行を加えなければなりません。あるいは、

```
UsePPosition "non-zero"
```

としておけば、ウィンドウの位置を (0,0) と指定すると、open されてもすぐにはマップされず、マウスによって初期位置を決めることができます。

11 問題点

バグもあるかもしれませんが、あったら教えて下さい。

索引

Graph クラスの関数

- axis, 8
- axis_box, 8
- child, 5
- circle, 9
- clear, 9
- color, 6, 9
- dash_line, 9
- EPSAppend, 10
- EPSOut, 10
- flush, 9
- forget, 9
- forgetLastPoint, 9
- full, 9
- full_circle, 9
- line, 9
- mouse_press, 9
- mouse_wait, 9
- open, 5
- pset, 9
- rectangle, 9
- resetPS, 10
- set_axis_with_digit, 8
- set_dash_line, 9
- set_double_dash_line, 9
- set_ratio, 7
- set_solid_line, 9
- set_xscale_full, 8
- set_yscale_full, 8
- text, 9
- xaxis, 8
- yaxis, 8

JWindow クラスの関数

- addLeftMargin, 7
- clear, 10
- depend_wm, 5
- EPSOut, 10
- expose, 10
- flush, 10
- get_index, 10
- get_window, 10
- graph, 6
- map, 6, 10
- mouse_press, 10

- mouse_wait, 10

- open, 4

- unmap, 10

Menu クラスの関数

- act, 11
- addClose, 11
- addExit, 11
- addGetString, 12
- addGetValue, 12
- addItem, 11
- addSubmenu, 11
- close, 11
- map, 11
- open, 11
- setStyle, 11
- setTitle, 11
- setType, 11
- unmap, 11

XController クラスの関数

- close, 10
- expose, 10
- flush, 10
- mouse_press, 10
- sync, 10