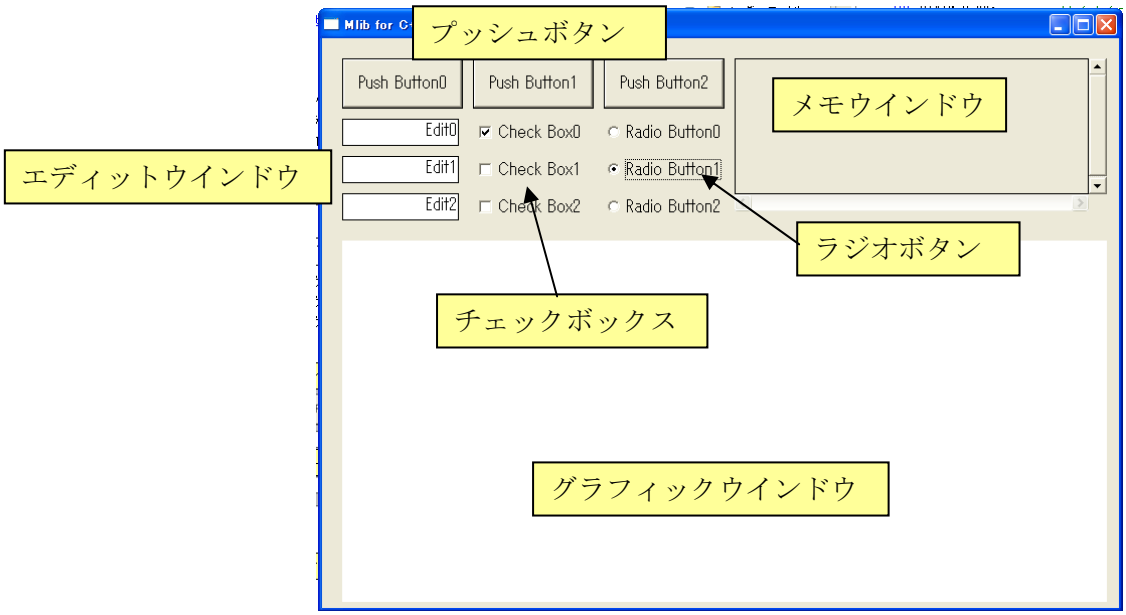
Win32 API based GUI Library for C Language (Ver 3.8)

ウインドウズＡＰＩを使用したＣ言語プログラムライブラリ　(MS Visual C 用)

本プログラムは WIN32API を使用した GUI によるＣ言語プログラムが可能なライブラリであり、画面出力、キー入力、関数のグラフ化などの基本機能を備えている。マイクロソフトの Visual studio Community Edition での使用が前提となるが、古い edition でも動作する。

・ ライブラリの説明

本ライブラリは WINDOWS OS の持つウインドウ関連の標準関数である WIN32 API を利用して、数値シミュレーションや計測器等の制御､取り込みプログラムなどの作成の基になるウインドウ入出力関連の作業を行うためのライブラリである。GUI をライブラリ内で作成するため、統合開発環境を用いなくとも、ウインドウズ対応のコンパイラと WIN32API を使用可能にする PlatformSDK(最新の Visual Studio ではインストール済み)があれば、どんな環境でも使用可能である。ANSI C に対応しており、C 言語の基礎を習ってはいるが、統合開発環境での GUI 作成に慣れてない人や C++などのオブジェクト指向言語はめんどくさいという人のためのライブラリである。研究室配属されたばかりの学生さんなどには最適である。本ライブラリは以下の６種類のウインドウ、ボタンを



持つメインウインドウを生成する。

・ライブラリの使用法

本ライブラリはウインドウや関数の初期値を定めた def.h と、関数群ライブラリの Mlib.h で構成される。使用法としては、プロジェクトのヘッダーファイルにライブラリを登録し、メインプログラムを新規作成する。メインプログラムには def.h, Mlib.h の順でインクルードして、**main**()関数にプッシュボタンが押されたときの処理を書くだけでよい．（Visual studiocommunity edition でのプロジェクト作成方法は http://miwalab.ei.st.gunma-u.ac.jp/mlib.html を参照）。

例）

```
#include "def.h"
#include "Mlib.h"
int main (int Number){
        Printf( “Hello World!” );
```

```
}
```

ウインドウ初期設定関連(def.h)

def.h の定数 MAINWIN_W と定数 MAINWIN_H はメインウインドウの横幅、縦幅であり、変更するとグラフィックウインドウのサイズ、メモウインドウのサイズを適宜調整して表示される。

・プッシュボタンは対応するボタンを押すことにより、main()関数で指定したプログラムが実行される。押されたボタンの番号に対応する整数値が仮引数 Number に入っているため、関数内で switch 文等で場合分けすればよい。定数 PB_NUM により配置するボタンの数が変更できる。ボタンの位置、サイズ、表示される文字は component()関数内でグローバル変数の sPB[]構造体を変更すればよい。

・チェックボックスは状態の ON、OFF を変更できるボックスであり、クリックごとに ON,OFF を切り替える。この状態は構造変数 sCK[チェックボックス番号].chk に反映され ON のとき 1、OFF のとき 0 になっている。定数 CK_NUM により配置するチェックボックスの数が変更できる。ボックスの位置、サイズ、表示される文字は component()関数内でグローバル変数の sCK[]構造体を変更すればよい。

・ラジオボタンはグループ化されたラジオボタンの中から一つを選択するボタンであり、クリックごとに ON,OFF を切り替える。この状態は構造体変数 sRD[ラジオボタン番号].chk に反映され ON のとき 1、OFF のとき 0 になっている。定数 RD_NUM により配置するラジオボタンの数が変更できる。ボックスの位置、サイズ、表示される文字は component()関数内でグローバル変数の sRD[]構造体を変更すればよい。

ラジオボタンのグループ化は sRD[].chk の初期値で設定する。以下の指定ではラジオボタンの数が 5 個のとき ID=0,1 と 2,3,4 がそれぞれグループになる。

```
sRD[0].chk=TRUE;
sRD[2].chk=TRUE;
```

・エディットウインドウは簡単な文や整数値などを実行前に入力したり、ちょっとした結果を出力するのに使用する。Get_int()関数等で指定したエディットウインドウから数値、テキストを取得できる. Set_text()関数等で文字列の出力が可能である。定数 ED_NUM により配置するボタンの数が変更できる。ボタンの位置、サイズ、表示される文字は component()関数内でグローバル変数の sED[]構造体を変更すればよい。

・メモウインドウは変数やコメント等をプログラム中から表示することが可能でありスクロールさせることができるため、多数の表示が必要なときに使用する。Print(TEXT("表示文字列"),変数等...)関数を使用する。

1 行の文字数は定数 BUFSIZE_W で指定している。記憶できる行数は定数 BUFSIZE_H で指定し、メモウインドウの表示行数は定数 DISPSIZE_H で指定する。ウインドウのサイズ、位置は component()関数内でグローバル変数の sME 構造体を変更すればよい。

・グラフィックウインドウは図が描画されるウインドウである。標準では定数 MAINWIN_W と定数 MAINWIN_H を変更すればこのサイズも自動的に変更される。グラフィックウインドウには Set_figure()関数により、複数のフィギュアウインドウを配置可能である。各フィギュアウインドウの左、右、上、下マージンは global 変数 MERGIN_L,R,T,B で変更できる。特にｙ軸の数値ラベルが幅広くなったときなど、適宜 MERGIN_L を大きくする必要がある。Plot_1d(),Plot_2d()関数でグラフを作成することができる

また、グラフィックウインドウに表示される文字のフォントの変更は、以下のように global 変数を変更する。

```
TCHAR Used_Font[32]=L"MS PGOTHIC";          //フォントをＭＳ　Ｐゴシックに設定する
Used_Font_Size=20;                          //フォントサイズを20に設定する
Italic_Font_Flug=1;                         //イタリック体に設定する
```

これらの変数はプログラム中で変更しても、その後の描画にすぐ反映される。

・スタティックは上記以外の文字のみを表示する. 定数ST_NUMにより配置するボタンの数が変更できる。ボタンの位置、サイズ、表示される文字はcomponent()関数内でグローバル変数のsST[]構造体を変更すればよい。

その他、多くの global 変数を定義しているので、各プログラム中で再定義しないように注意する。

関数

（メモウインドウ関連）

void Printf(char *fmt, ...)

メモウインドウに書式付文字列表示を行う。使用例は printf 文と全く同じ書式指定が可能。
Printf(“Print Hello!¥n”), Printf(“x=%d y=%f¥n” x,y)等

void Print(TCHAR *fmt, ...)

メモウインドウに簡易書式付文字列表示を行う。引数が TCHAR 型であることに注意。ワイド文字限定であれば、文字列の前に L プレフィクスを用い、TCHAR 型であれば TEXT(“”)マクロ内に文字列を記述する。使用例は printf 文とほぼ同じ。使用変数は整数型と実数型のみ。変数を表示する場合はそれぞれ%d、%g で表示位置を指定し、文字列の後ろにコンマで変数名を記述する。改行文字は ‘¥n’

void inputbuf(TCHAR mess[hbuf],int crflug)

メモウインドウに TCHAR 型の文字配列 mess の内容を表示する。
crflug=0 以外のとき改行する。

（グラフィックウインドウ関連）

グラフィックウインドウは独立した座標を持ち、左上が０で、ピクセル単位で座標を指定する。

void Plot_pen(int pf, int pw, int pc)

グラフィックウインドウに描画する色を指定する。
pf: 0-実線　1-破線　2-点線　3-１点鎖線　4-２点鎖線　5-描画しない
pw: ペンの太さをピクセル単位で指定。 0 は１ピクセル
pc: 0-黒　1-赤　2-緑　3-青　4-黄色　5-マゼンダ　6-シアン　7-白
8 以上を指定すれば Plot_Pen()を呼び出す毎に自動的に色を 0～6 まで巡回させる。

void Line(int x1,int y1,int x2,int y2)

グラフィックウインドウに直線を引く
x1,y1 は始点の座標、x2,y2 は終点の座標
色、線の種類は Plot_pen()で指定

void Rect(int x1,int y1,int x2,int y2,int bfflug)

グラフィックウインドウに四角形を描く、色、線の種類は Plot_pen()で指定
x1,y1 は左上の座標、x2,y2 は右下の座標

bfflug=0	枠のみ。枠内は描画しない。
bfflug=1	枠と枠内を Plot_pen()で指定した色で塗りつぶす
bfflug=それ以外	枠と枠内を白で塗りつぶす

void Circle(int x1,int y1,int x2,int y2,int bfflug)

グラフィックウインドウに楕円を描く、色、線の種類は Plot_pen()で指定
X1,y1 は左上の座標、x2,y2 は右下の座標を指定し、その四角形に接する楕円を描く。
bfflug=0　枠のみ枠内は描画しない。
bfflug=1　枠内を Plot_pen()で指定した色で塗りつぶす
bfflug=それ以外　枠のみを白で塗りつぶす

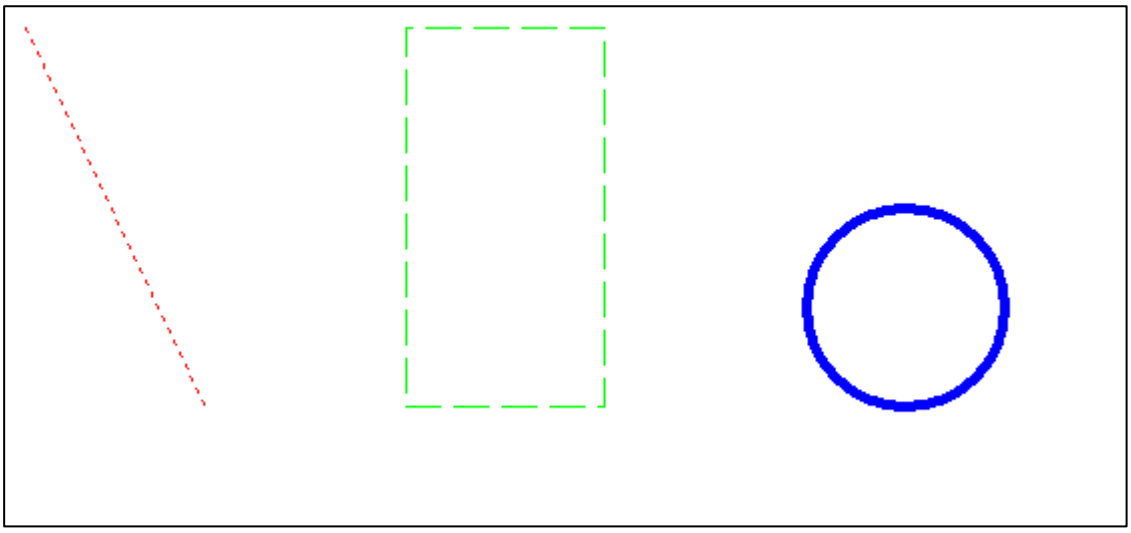
void Fig2clipboad()

グラフィックウインドウ全体をクリップボードにコピーする。

使用例）

```
int main(int Number){
        Plot_pen(2,1,1);          //ペンの指定。（点線、太さ１、赤）
        Line(10,10,100,200);      //直線の描画
        Plot_pen(1,2,2);          //ペンの指定。（破線、太さ２、緑）
        Rect(200,10,300,200,2);   //四角の枠のみの描画
        Plot_pen(0,5,3);          //ペンの指定。（実線、太さ５、青）
        Circle(400,100,500,200,1);//円の描画
        Fig2clipboard();          //クリップボードにコピー
}
```

実行後、Word 等に図をペーストすれば以下の図が得られる。（実際は黒枠はない）



（フィギュアウインドウ関連）

フィギュアウインドウ ―― グラフィックウインドウ内にあるグラフ作成用の仮想ウインドウで一つのフィギュアウインドウには一つのグラフを描くことができる。

```
void Clf(int clflug)
```

フィギュアウインドウの消去。

clflug=-1	現在のフィギュアウインドウの枠だけ描画
clflug=0	現在のフィギュアウインドウの枠を描画し、枠内を消去
clflug=1	現在のフィギュアウインドウをラベルも含めて消去、最大最小の軸情報もクリア
clflug=2	グラフィックウインドウを全て消去、全てのフィギュアウインドウの最大最小の軸情報もクリア

```
void Set_figure(int i1,int i2,int n)
```

グラフィックウインドウを i1,i2 で構成されるマトリクス内の n で指定される位置に現在のフィギュアウインドウを作成し、カレントのフィギュアウインドウに設定する。以後の Plot1d(…)等の描画はカレントフィギュアウインドウに描かれる。一旦、指定したウインドウには黒枠のみが描かれる。

i1: ウインドウの縦方向の数
i2: ウインドウの横方向の数
n:指定するウインドウの番号、左上を１番とし右に数えて行く

Set_figure(…)を行った順番にフィギュア IDが割り当てられる。この数値はglobal変数のiID_Cfwに入っている。初めて指定したときはフィギュアウインドウの最大最小値がクリアされる。二回目以降に指定したときは前に指定した同じフィギュアウインドウの設定値が用いられるため、最大値最小値の値は記憶されている。1回指定すれば再指定する必要はない。一方、一度指定したフィギュアウインドウに違う軸を持つ図を描画する場合はclf(1)で一旦図を消去して、再指定する必要がある。

void Axis_xcap(double min,double max, char *label)

ｘ軸の最小、最大設定、数値ラベル、キャプションを表示する。

min: ｘ軸の最小値。（描画する図の最初のデータに対応するｘ軸の数値を入れる）

max: ｘ軸の最大値。（描画する図の最後のデータに対応するｘ軸の数値を入れる）

label: ｘ軸キャプションの文字列（100文字以内）の先頭アドレス

数値ラベルはminからmaxの間の適当な数値を自動的に計算して描画する。未定のときはmin=max=0を指定すれば、キャプションのみを描画する。このときはどこで呼び出しても良く、軸ラベルはPlotxyにおいて自動的に作成される。Plot1dではｘ軸数値ラベルは自動生成されない。

void Axis_ycap(double min,double max, char *label)

ｙ軸の最小、最大設定、数値ラベル、キャプションを表示する。

min, max ： ｙ軸の最小値、最大値

label: ｙ軸キャプションの文字列（100文字以内）の先頭アドレス

数値ラベルはminからmaxの間の適当な数値を自動的に計算して描画する。軸スケールがあらかじめ決まっているときはPlot1d関数より先に呼び出す。未定のときはmin=max=0を指定すれば、キャプションのみを描画する。このときはどこで呼び出しても良く、軸ラベルはPlot1d等において自動的に作成される。数値ラベル幅が長いとき、キャプションがウインドウ枠を飛び出すことがある。このときは def.h 中のMERGIN_Lを大きくするとよい。

void Plot1d_int(int *yn, int n)

現在Set_figure(…)で指定されているフィギュアウインドウに整数型配列ynのグラフを描画。
色、線の種類はPlot_pen(…)で指定。Axis_ycap(…)で軸の最大最小が指定されていないときは、自動的に配列内の最大値最小値を探しｙ軸を設定、描画する。

yn ： 整数型の配列

n ： 表示する配列の最大要素

表示したい配列の要素数が定義された要素数を超えると、無意味なデータが表示されるため注意が必要。nが定義よりも小さければ問題ない。Clf(1)もしくはClf(2)関数で図のクリアをしなければ、同じウインドウに複数(<20)のグラフを描画できる。

void Plot1d (double *yn ,int n)

指定されているフィギュアウインドウにdouble型配列ynのグラフを描画。
色、線の種類はPlot_pen()で指定。Axis_ycap(…)で軸の最大最小が指定されていないときは、自動的に配列内の最大値最小値を探しｙ軸を設定、描画する。

yn :doubleの配列

n ： 表示する配列の最大要素

表示したい配列の要素数が定義された要素数を超えると、無意味なデータが表示されるため注意が必要。nが定義よりも小さければ問題ない。Clf(1)もしくはClf(2)関数で図のクリアをしなければ、同じウイン

ドウに複数(<20)のグラフを描画できる。

void Plotxy (double *yn , double *yn ,int n)

指定されているフィギュアウインドウに double 型配列 xn を x 軸、double 型配列 yn を y 軸に対応させたグラフを描画。色、線の種類は Plot_pen()で指定。Axis_ycap(…), Axis_xcap(…)で軸の最大最小が指定されていないときは、自動的に配列内の最大値最小値を探し y 軸を設定、描画する。

xn, yn :double の配列

n : 表示する配列の最大要素

表示したい配列の要素数が定義された要素数を超えると、無意味なデータが表示されるため注意が必要。n が定義よりも小さければ問題ない。Clf(1)もしくは Clf(2)関数で図のクリアをしなければ、同じウインドウに複数(<20)のグラフを描画できる。

void Aspect_ratio(double ax, double ay)

フィギュアウインドウのサイズを(幅：高さ)＝(ax:ay)で表される比率に変更する。プロット関数より前に指定する必要がある。

標準ではグラフィックウインドウのサイズを Set_figure 関数で決まるウインドウ個数に従って均等に割り振られる。この関数より、標準サイズの縦横サイズを超えない範囲で指定した比率にサイズを定める。

void Grid_on(int grflug)

図にグリッドラインを描画する。数値ラベルの値のある座標に直線を引く

軸の最大最小を決めた（Plot()関数、Axis_ycap(), Axis_xcap()）後に指定する。

grflug=0 : y 軸の 0 レベルのみを引く

grflug=1 : y 軸のみグリッドライン描画

grflug=2 : x 軸のみグリッドライン描画

grflug=3 : x 軸、y 軸両方にグリッドライン描画

void Legend(char* text, int posflug)

Plot()関数で重ね書きされたグラフの数だけ凡例を表示する。凡例 text は“line1|line2|line3”のように表記する。線の名前の区切り文字は'|'である。

posflug=0 : 枠外右下に表示

posflug=1 : 枠内左上に表示

posflug=2 : 枠内右上に表示

posflug=3 : 枠内左下に表示

posflug=4 : 枠内右下に表示

void Text_draw(double x , double y, char* text);

フィギュアウインドウの各軸の値を基準にした指定座標(x,y)に文字textを表示する。グラフのタイトル、凡例に表記できない情報等の記述に使用できる。フォントの変更は Used_Font, Used_Font_Size等のグローバル変数を変更すればよい

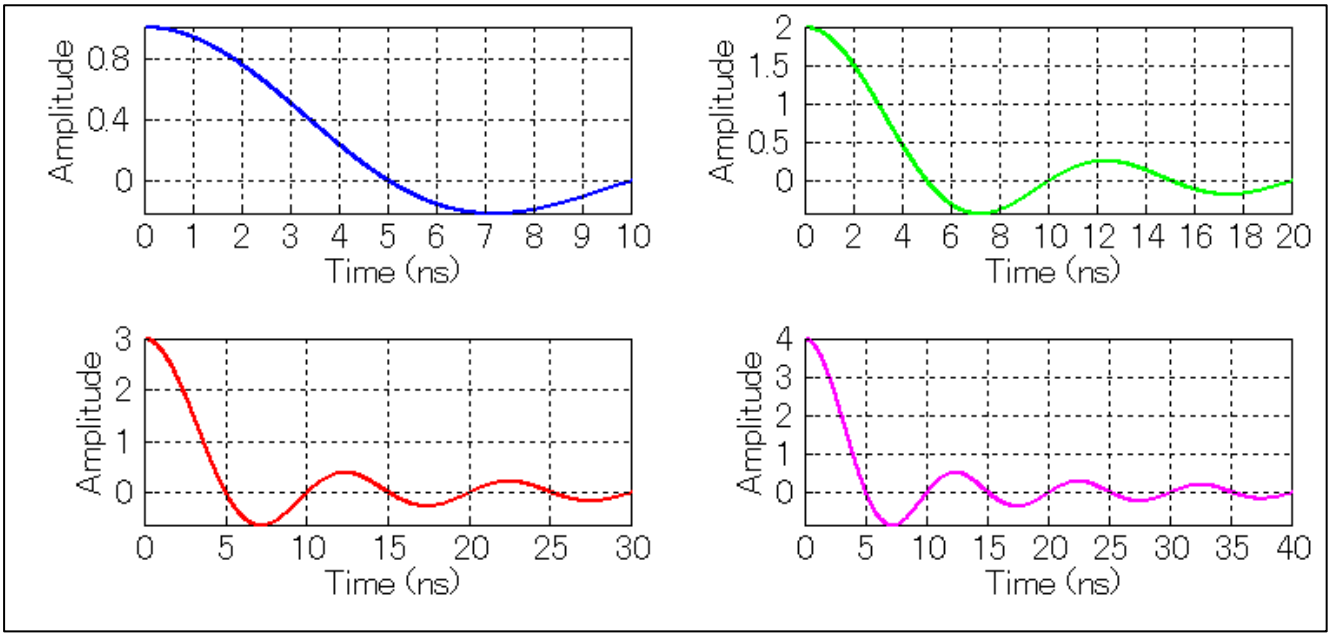
void Refresh()

強制的に画面の再描画を行う。描画系関数は main()関数が終了しなければ画面に反映されない。main()関数実行中に描画する場合に

使用例)

```
int main(int Number){
        double x,y[1000],pi=3.1415926535;
        int N=1000,i,j,k,s[1000];
```

```
        Clf(2);                         //グラフィックウインドウ全体を消去

        for(j=1;j<=4;j++){
                Set_figure(2,2,j);      //グラフィックウインドウを2*2のフィギュアウインドウに分割
                Plot_pen(0, 2, 8);      //ペンを実線、太さ2ピクセル、色の自動循環に指定
                for(i=0;i<N;i++){
                        x=i/(double)N;
                        y[i]=(sin(2*pi*(j*x)+1e-10)/(2*pi*(j*x)+1e-10))*j;
                }

                Plot1d(y,N);//グラフの描画。Axis_ycapにより軸の最大最小を指定する前に
                            //Plot関数を使っているためｙ軸の最大最小を自動計算し描画する
                Axis_ycap(0,0,"Amplitude");  //軸が書かれているためキャプションのみ描画
                Axis_xcap(0,10*j,"Time (ns)");  //ｘ軸は常に最大最小を指定する。
                Grid_on(XYGRID_LINE);
        }
        Fig2clipboard();            //グラフィックウインドウをクリップボードへ
}
```



Plot2d(double *yn, int nx, int ny, double cmin, double cmax, int plflug, int cbflug, int szflug)

指定されているフィギュアウインドウに double 型配列 yn の 2D 画像を描画。

配列 yn は二次元データを 1 次元配列に並び替えたデータとして与える。x 方向、y 方向のポイント数は nx, ny で与えられ、

0 番目	x=0;y=0
1 番目	x=1;y=0
…	
nx-2 番目	x=nx-2;y=0
nx-1 番目	x=nx-1;y=0
nx 番目	x=0;y=1
nx+1 番目	x=1;y=1
…	
nx*ny-2 番目	x=nx-2;y=nx-1
nx*ny-1 番目	x=nx-1;y=ny-1

の順番で二次元データを 1 次元配列 yn に格納する。

cmin,cmax: カラースケールの最小、最大値の指定、両方とも０のとき最大最小値を自動的に探す。
plflug: ０のときカラーバー表示 off、１のときカラーバー表示
cbflug: カラーマップの指定。０のときグレースケール、１のとき虹色、２のとき青、赤の循環色
szflug: ０のとき画像サイズに関係なくフィギュアウインドウの規定サイズに拡大して描画
（データの縦横サイズがウインドウサイズより大きいと画像が乱れるので注意）
１のときフィギュアウインドウのサイズを正方形にして描画
２のとき、データのサイズに対応したピクセル数で描画（最もきれい）

使用例）

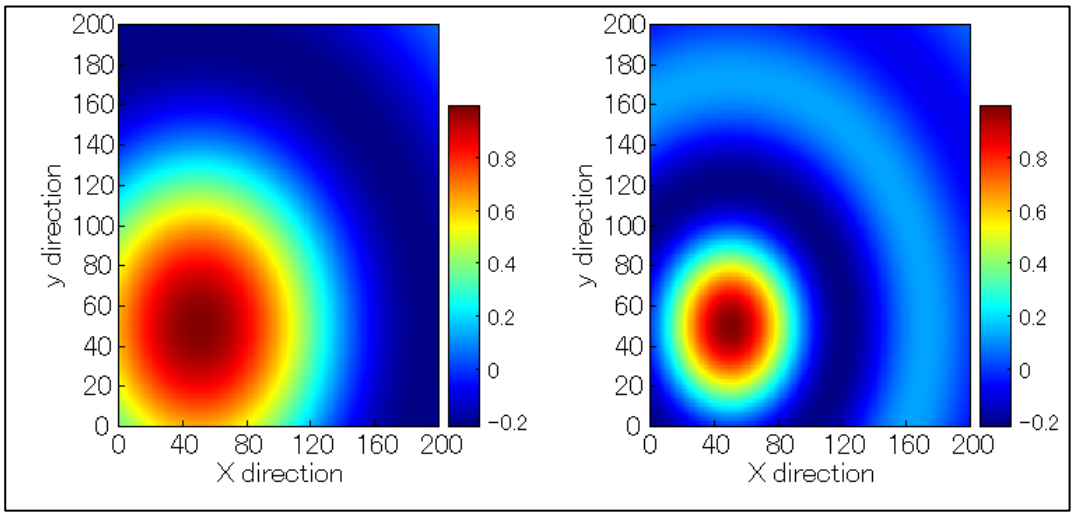
```
int main(int Number){
      int i,j;
      int nx,ny,xm=200,ym=200;
      double x[200*200],r;
      double pi=3.1415926535;

         for(j=1;j<=2;j++){
              for(nx=0;nx<xm;nx++){
                   for(ny=0;ny<ym;ny++){
                        i=ny*xm+nx;  //二次元データを1次元データにするための変換
                        r=2*pi*sqrt(pow(nx-50+.5,2)+pow(ny-50+.5,2))/200*j;
                        x[i]=sin(r)/r;
                   }
              }

              Set_figure(1,2,j);    //グラフィックウインドウを縦1個、横2個のフィギュアウインドウに分割
              Plot2d(x,xm,ym,0,0,RAINBOW,LEGEND_ON,SIZE_NORMAL); //二次元データ描画
              Axis_ycap(0,ym,"y direction");
              Axis_xcap(0,xm,"X direction");
         }
         Fig2clipboard();
}
```

Szflug による画像の違い

200*200 のデータを作成し、描画している。Plot2D ではカラースケールの最大最小を０(自動計算)、カラーマップを虹色、カラーバーを表示、画像サイズはノーマルとしている。

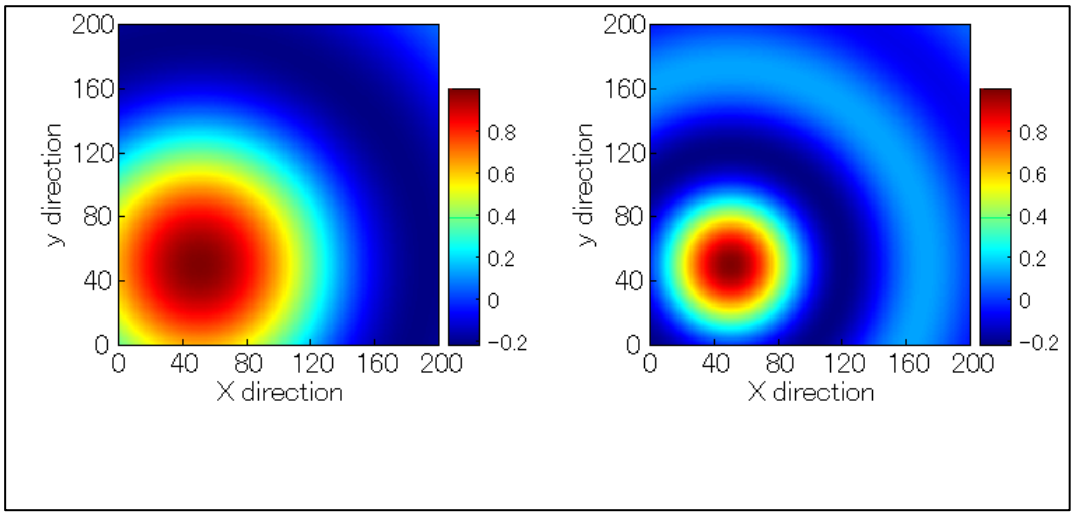
szflug=0　(SIZE_NORMAL)

画像データが、指定したウインドウサイズに拡大されて表示されるため、縦横比が異なっている。



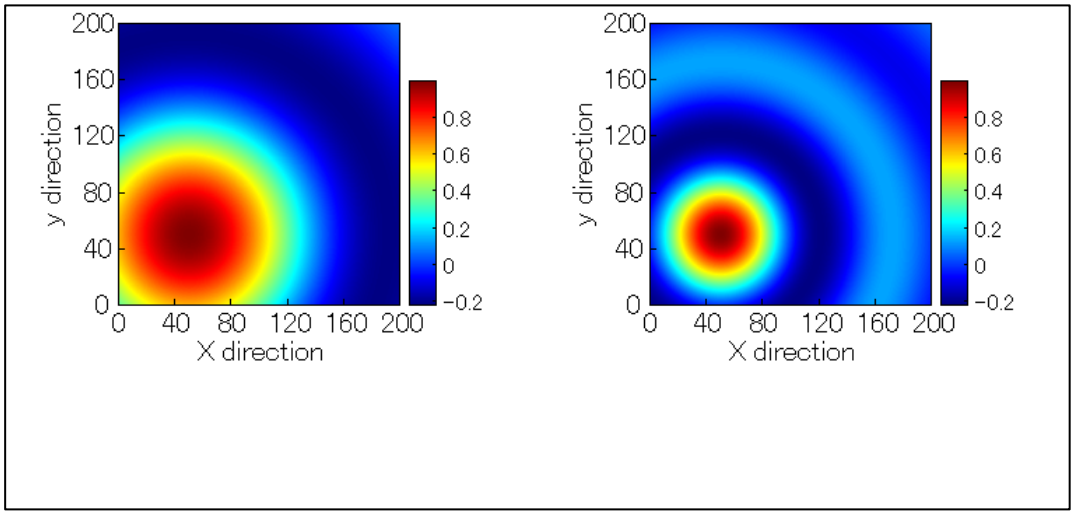
szflug=1　(SIZE_SQUARE)

描画ウインドウが正方形になるように変更され、拡大されて表示される。イメージが荒くなる



szflug=2　(SIZE_DATA)

描画ウインドウがデータのサイズと同じピクセルサイズに変更され､そのまま表示される｡もっともきれい。



（キーボード入力関連、その他）

int Input_int(),　double Input_double(),　TCHAR *Input_text(),　char *Input_char()

ダイアログボックスを表示し、それぞれ整数値、実数値、テキストを入力し、戻り値として返す。数字入力では数値以外を入力すると 0 が返される。文字入力では戻り値はなしで、引数に文字列の先頭アドレスを返す変数を指定する。単なるキー入力待ちとしても使用できる。

int Get_int(int i),　double Get_double(int i),
void Get_text(int i, TCHAR *Buffer) , void Get_char(int i, char *Buffer)

エディットウインドウから、それぞれ整数値、実数値、テキストを読み取り、戻り値として返す。数字入力では数値以外を入力すると 0 が返される。文字入力では戻り値はなしで、引数に文字列の先頭アドレスを返す変数を指定する。i: はエディットウインドウの番号(0 以上)

Void Set_double(int i, double x), void Set_text(int i, TCHAR *buffer), void Set_char(int i, char *buffer)

エディットウインドウに実数値、もしくは*buffer で指定したワイド文字及びマルチバイト文字列を書き

込む。i: はエディットウインドウの番号(0 以上)

void Pause()

一時停止のダイアログボックスを表示

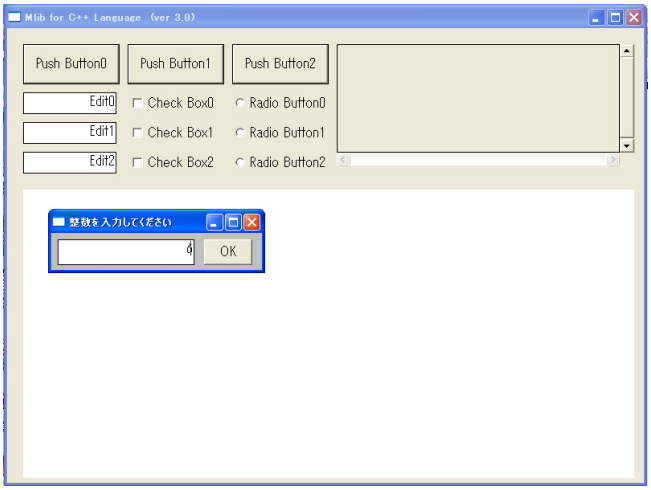
void Delay(int msec)

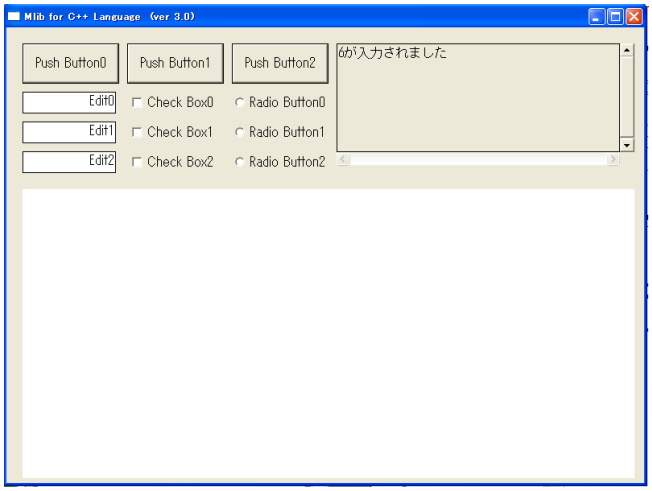
待ち時間をミリ秒で指定

使用例)

```
int main(int Number){
        int i;
        i=Input_int();
}
```

以下のようなダイアログが出て入力待ち状態になる。
入力後右図のようにメモウインドウに入力値が表示される。





変更履歴

Ver3.7->3.8

・グラフ系関数の強制描画を行う **Refresh()**関数追加
・Visual Studio Community Edition での動作確認、エラー、警告の修正
・main 関数の出力を void 型から int 型に変更

Ver 3.5→3.7

- Plot 関数の凡例を作成する **Legend()**関数追加
- フィギュアウインドウに文字を描く **Text_draw()**関数追加
- Printf,Print 関数での文字表示において表示結果がリアルタイムに現れるように修正
- エディットウインドウに double 型の数値を出力する **Set_double()**関数の追加
- 関数のプロトタイプ宣言化
- 文字列処理関連のバグフィクス

Ver 3.3→3.5

- ｘ軸とｙ軸に任意の数値データを指定できる **Plotxy()**関数を追加
- フィギュアウインドウのサイズを変更する **Aspect_ratio()**関数を追加

Ver 3.1→3.3

変更概要は主に、関数への文字列入力を TCHAR 型ではなく char 型で入力できるように変更した。

- char 型の出力関数 **Prinf()**の追加
- Print 関数のアルゴリズム修正、改行文字の “¥¥n”から”¥n” への修正
- Axis_ycap,Axis_xcap 関数のキャプション文字列の型を char 型のみに限定
- エディットウインドウから文字列を取り出す Get_text 関数において、文字列格納アドレスの戻り値を返すのではなく、引数に文字列格納アドレスを指定するように変更。
- エディットウインドウから char 型文字列を取り出す **Get_char()**関数の追加。
- 入力ダイアログへの入力を char 型の文字列として取り出す **Input_char()**関数の追加
- エディットウインドウに char 型の文字列を出力する **Set_char()**関数の追加

Ver 2.6→3.1

- ラジオボタン、チェックボックスを使えるように追加
- Print_text 関数の名前を **Print()**関数に変更