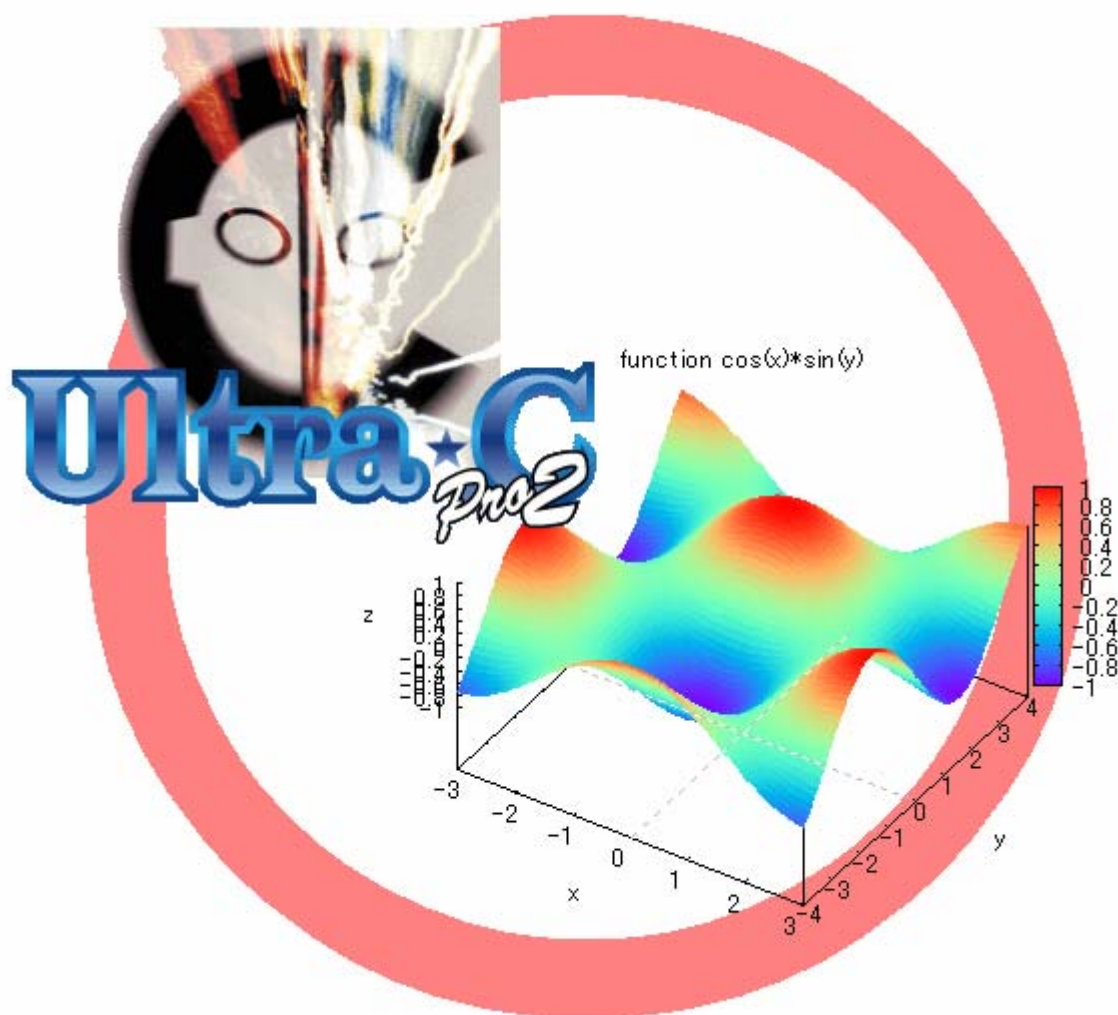


Ch Professional 6.0

Ultra-C Pro バージョン 2.1 からの移行ガイド



株式会社 ラネクシー®

Run in the Next!

内 容

1. はじめに.....	1
2. ユーザーインターフェース	2
2.1 起動方法	2
2.2 ツールバー	2
ツールバー	2
デバッグバー	2
2.3 ウィンドウ.....	2
編集ウィンドウ.....	3
デバッグウィンドウ.....	3
アウトプットウィンドウ	3
Debug Console Window (デバッグコンソールウィンドウ)	5
グラフィックの出力先.....	5
2.4 ソースファイルの保存.....	5
2.5 デバッグ方法.....	6
3. ライブラリ関数の互換性	8
3.1 概要.....	8
「ライブラリ関数」	8
コンソールI/O関数.....	8
ディレクトリ操作関数.....	8
グラフィック関数.....	8
I/Oポート関数.....	10
3.2 サンプルプログラムの互換性	10
3.3 グラフィックプログラムの変更例.....	13

1. はじめに

本ガイドは、Ultra-C Pro バージョン 2.1 をご使用のお客様が弊社製品 Ch Professional 6.0 にスムーズに移行できるようにすることを目的にしています。

本ガイドでは主に、Ultra-C Pro バージョン 2.1（以下単に Ultra-C Pro）と Ch Professional 6.0（以下単に Ch Professional）との相違点について記述しています。Ch Professional 自体の仕様、リファレンス情報については、以下のマニュアルをご覧ください。

- Ch 言語環境 バージョン 6.0 ユーザーズガイド - ファイル名 : chguide.pdf
Ch の言語機能について記載しています。Ch の文法を修得し、Ch を使って「何ができるか」をお調べになりたい場合は、このガイドをご覧ください。本ガイドでは、これ以降『ユーザーズガイド』と称します。
なお、パッケージ版にはこのガイドの製本版も同梱されています。
- Ch 言語環境 バージョン 6.0 リファレンスガイド - ファイル名 : chref.pdf
Ch で使用可能な関数、クラスおよびコマンドの詳細なリファレンス情報を提供します。Ch を用いてのプログラミングの際に参照してください。本ガイドでは、これ以降『リファレンスガイド』と称します。
なお、日本語版では Ultra-C Pro で使用されている C 言語の標準ライブラリ関数に関する部分のみの情報が提供されています。本ガイドのあとのセクションで言及されているプロット機能、C++のクラス、C99 Standard の仕様（計算配列など）に関するリファレンス情報については、申し訳ございませんが、製品インストール時にコピーされる同名の英語版マニュアルをご覧ください。
- Ch IDE および Ch コマンドシェル入門 - ファイル名 : chide.pdf
Ch Professional に同梱されている統合化開発環境（IDE）Ch IDE に関する入門情報を提供し、これを用いて簡単なプログラムのデバッグ方法について紹介します。さらに、コマンドラインインターフェース環境の Ch コマンドシェルの使い方についても述べます。本ガイドでは、これ以降『IDE ガイド』と称します。
- Ch IDE ユーザーズガイド - ヘルプファイル
Ch IDE のヘルプ（HTML ファイル）です。Ch IDE の [ヘルプ] メニューから [ヘルプ] を選択し、[Japanese] のリンクをクリックすると開きます。

2. ユーザインターフェース

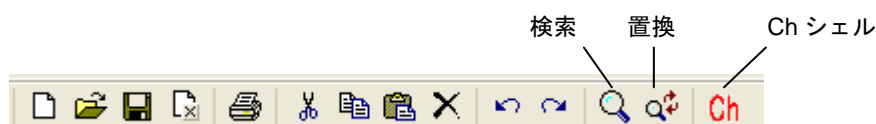
2.1 起動方法

[スタート] メニューから、[すべてのプログラム] → [SoftIntegration Ch 6.0 Professional] を選択します。Ultra-C Pro のようなグラフィカルなユーザインターフェースで Ch Professional をご使用になりたい場合は、ここで [Ch IDE] をクリックします。本ガイドでは、これ以降このインターフェース環境 (Ch IDE) で製品を使用していることを前提とします。

Ch Professional では、Ch シェルというコマンドプロンプトに似たコマンドライン環境で製品を使用することも可能です。この環境では Ch プログラムのソースファイルを指定して実行できるだけでなく、Ch 言語の 1 行 1 行を対話形式で実行することもできます。Ch シェルについては、『ユーザズガイド』の「移植可能な対話型コマンドシェルとシェルプログラミング」、または『IDE ガイド』の「Ch コマンドシェル入門」をご覧ください。

2.2 ツールバー

ツールバー

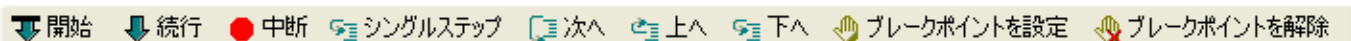


Ch IDE 独自のツールバーボタンは、以下のとおりです。

- [検索] 編集ウィンドウ内で文字列の検索を行います。Ultra-C Pro の [文字列の検索] コマンドに相当します
- [置換] 編集ウィンドウ内で文字列の置換を行います。Ultra-C Pro の [文字列の置換] コマンドに相当します
- [Ch シェル] Ch Professional をコマンドライン環境で実行する Ch シェルを起動します

デバッグバー

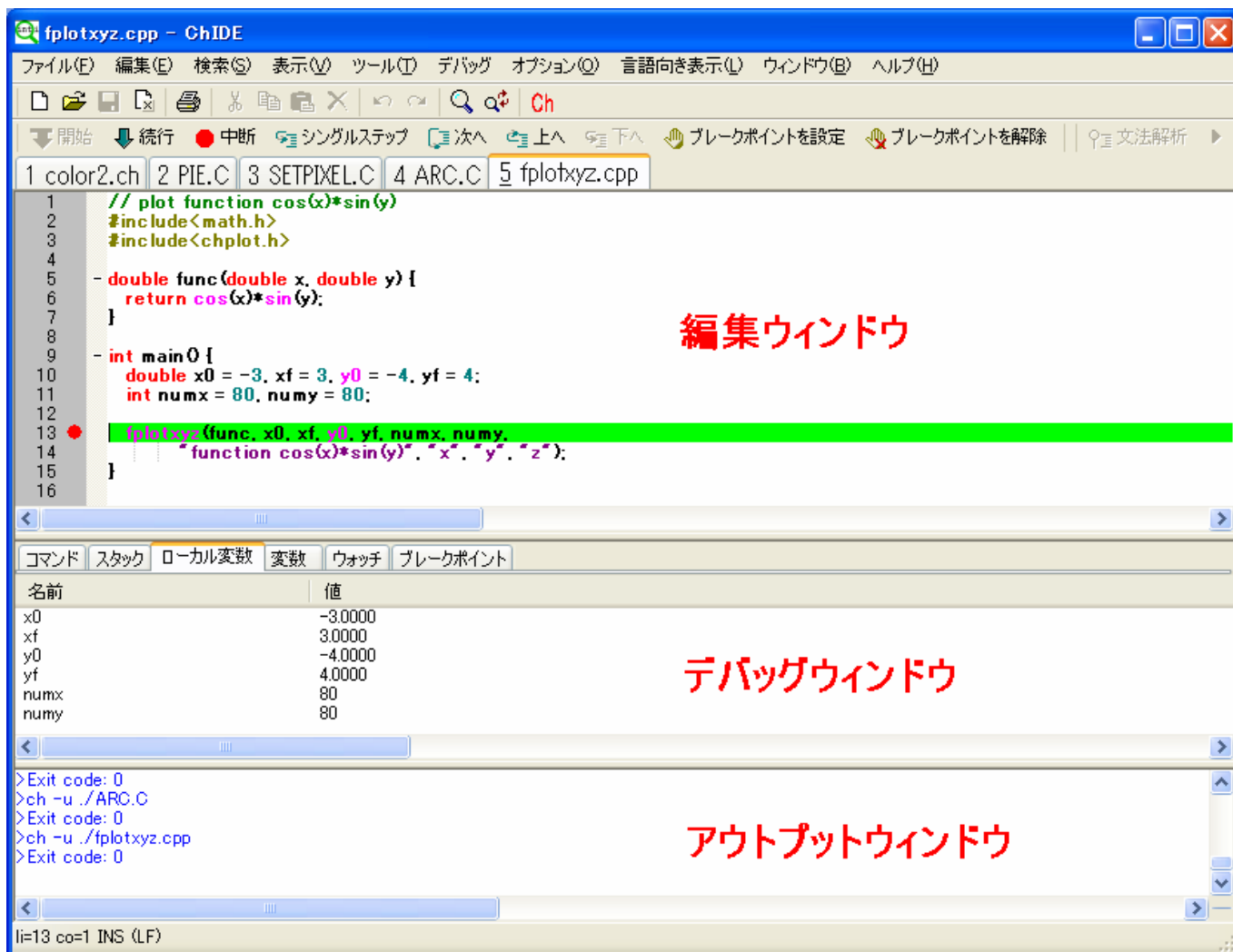
Ch IDE では、デバッグモードで以下のボタンが使用可能です。



これらのボタンの機能については、「[デバッグ方法](#)」のセクションで説明されます。

2.3 ウィンドウ

Ch IDE を起動すると、以下の図のようなウィンドウが表示されます。



ここでは主として、Ultra-C Pro との相違点について説明します。それぞれのウィンドウの詳細については、『IDE ガイド』をご覧ください。

編集ウィンドウ

Ultra-C Pro の [プログラム] ウィンドウに相当します。Ch IDE では、C (C++) 言語のキーワードおよび書式に合わせて色分けされて表示されます。

デバッグウィンドウ

上図のように、複数のタブから選択して表示を切り換えます。デバッグコマンドの入力とデバッグモードでの参照情報を出力します。Ultra-C Pro の [式の参照] ウィンドウの機能は、ここに含まれていません。

アウトプットウィンドウ

プログラムの実行による文字入出力とメッセージ出力がこのウィンドウに対して行われます。Ultra-C Pro の [標準入出力] ウィンドウと [エラー情報] ウィンドウの機能を含んでいます。

プログラムの実行を試みたとき、または [文法解析] ボタンをクリックしたときにプログラムに不適切な構文が含まれていると、このウィンドウにエラーまたは警告メッセージが表示されます。これらのメッセージは、現行バージョンの Ch Professional ではすべて英語になっています。以下、主なメッセー

ジについて説明します。

ERROR: variable 'va/' not defined

変数 'va/' が定義されていません
多くの場合、スペルミスと思われる。

ERROR: command 'com' not found

コマンド 'com' が定義されていません
多くの場合、スペルミスと思われる。

ERROR: syntax error before or at line *linenum* in file *filename*

ソースファイル *filename* の *linenum* 行目またはその直前に文法エラーが見つかりました
『IDE ガイド』にあるように、この行は赤字で表示され、この行をダブルクリックすると、編集
ウィンドウ内の該当する行とともに黄色の背景で強調表示されます。さらにそのあとの 2 行で、
たとえば以下のようにエラーの原因と思われる部分が指摘されます。

```
==>: printf("Hello, world\n");  
BUG: printf("Hello, world\n"); <== ???
```

ERROR: cannot execute command 'filename'

コマンド '*filename*' を実行できません
このエラーは文法解析対象のソースファイルにエラーが発見された結果、実行できなかったこと
を示しています。したがってエラーが発見されたときの最後のメッセージ行として必ず表示され
ます。

ERROR: missing ';'

文の終わりを示すセミコロン ';' がありません

ERROR: missing '}'

ブロックの終わりを示す '}' がありません

ERROR: in program ending for file '*filename*', may be missing }

ソースファイル '*filename*' に対するプログラムが正しく終了していません。おそらくブロック
の終わりを示す '}' がいないためと思われます

**WARNING: missing return statement for function *func()* and default zero is used at line *linenum*
in file *filename***

警告: ソースファイル *filename* の *linenum* 行目で、関数 *func()* 用の return 文がないため、
デフォルトの 0 が返り値として使用されます。
このメッセージはエラーではなくて警告を示していますので、[実行] コマンドを選択した場合は
上記のような代替措置を施して実行が完了します。

ERROR: function '*func()*' not defined

関数 '*func()*' が定義されていません
多くの場合、スペルミスと思われる。

ERROR: number of argument is *num1*, need *num2* arguments

引数の数は *num1* 個となっていますが、実際には *num2* 個必要です
関数呼び出しの際に指定している引数の個数が、実際の関数定義と一致していません。

ERROR: cannot find file 'file'

ファイル 'file' が見つかりません

通常、プログラムの先頭でインクルードしているヘッダファイルが見つからないことを示しています。ヘッダファイル名のスペルミスか、あるいは Ultra-C Pro 専用のヘッダファイル 'igraph.h' のインクルード行がそのまま残っている場合などが考えられます。

ERROR: argument *arg* of the function is undefined or not a valid expression

関数の引数 *arg* が定義されていないか、または有効な式ではありません

指摘されている引数の型が、関数定義で指定されている型と一致していないと思われます。

Debug Console Window (デバッグコンソールウィンドウ)

このウィンドウは、デバッグモードに移行した際に表示されます。デバッグモードでの文字出力は、[アウトプットウィンドウ](#)ではなく、このウィンドウ上に出力されます。

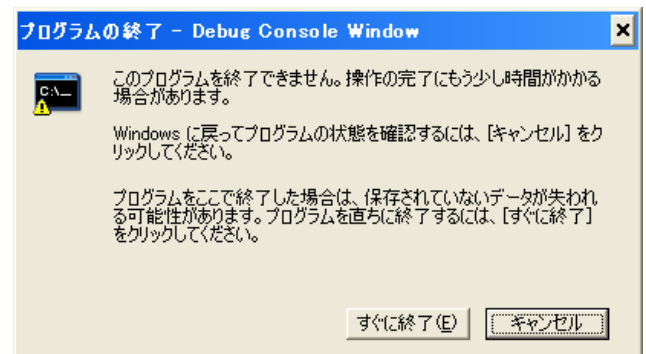
[表示] メニューで [Debug Console Window を常に最前面に] にチェックが入っている場合、この Debug Console Window は常に前面に表示されている状態になっています。他のアプリケーションウィンドウを見たい場合などにこの状態を解除したい場合には、このメニュー項目を選択して、このチェックをはずしてください。

ご注意：

Debug Console Window を、そのタイトルバーの右端の [閉じる] ボタン ('x' マーク) をクリックして直接閉じることはできません。そのようにしようとすると図のような警告メッセージが表示されます。ここでは、[キャンセル] をクリックして元の状態に戻ってください。[すぐに終了] をクリックすると、Ch IDE も強制終了し、保存していないソースや IDE のセッション情報はすべて失われてしまいます。

デバッグモードでないときに Debug Console Window を閉じたい場合は、Ch IDE の [表示] メ

ニューから [Debug Console Window] を選択して、このチェックをはずしてください。



グラフィックの出力先

Ch Professionalで二次元または三次元のプロットを含むプログラムを実行すると、独自のウィンドウが開きます。このウィンドウはUltra-C Proの [グラフィック] ウィンドウに機能的に対応するものですが、Ch Professionalの場合、Gnuplotと呼ばれるオープンソースの独立したアプリケーションを起動し、このウィンドウ上にプロット結果を出力します。Gnuplotについては、「[グラフィック関数](#)」のセクションでさらに説明します。

2.4 ソースファイルの保存

現行バージョンの Ch IDE では、ソースコードを編集したあとで [実行]、[文法解析]、[デバッグの開始] を選択すると、このソースファイルが、ユーザに操作を問い合わせることなく自動的に上書き保存されます。

また、新規に作成したプログラムに対してこれらの動作を行おうとすると、Ch IDE はこのプログラム

をソースファイルとして保存するための保存先とファイル名を問い合わせます。すなわち、**保存されていないプログラムは実行／解析／デバッグができません。**

結局、ソースファイルを閉じる際にファイルを保存するかどうかを Ch IDE から問い合わせられるのは、最後にソースコードを変更したのちにこれらの動作を一度も行わず、実際に未保存の個所が残されている場合のみです。何も問い合わせがないからといって、このソースが開いてから一切変更されていないことにはなりませんので、ご注意ください。

2.5 デバッグ方法

Ultra-C Pro と Ch Professional でのデバッグ機能、方法の主な違いは、以下のとおりです。

- Ch Professionalは、[デバッグウィンドウ](#)という複数のタブを含んだ独自のウィンドウでデバッグコマンドの指定、デバッグモードにおける各種情報の出力を行います。
- Ch Professional では、行指定だけでなく、関数、変数、条件を指定したブレークポイントの細かい設定が可能です。なお、Ultra-C Pro の「中断点」は、Ch Professional では**ブレークポイント**と呼ばれています。Ch Professional でも「中断点」という用語が使用されていますが、これは「**コールスタック** (call stack) ごとの中断点」、すなわち現関数ならば「**現在実行を中断している文**」、それ以外ならば「**現関数までの各関数の呼出し文**」のことを意味しますので、ご注意ください。
- Ch Professional では、デバッグモード中に各スタックごとのローカル変数を参照／変更することが可能です。
- Ch Professional には、トレース機能はありません。

Ch Professional も Ultra-C Pro のように、各デバッグコマンドをメニューやツールバーボタンを選択して実行することができますが、より細かいデバッグ操作を行うには、デバッグウィンドウの [コマンド] タブを選択して表示されるデバッグコマンドを使用します。これらのコマンドでは、メニューやツールバーボタンでは設定できない操作を、引数などを指定して行うことができます。

以下のデバッグコマンドの一覧は、デバッグウィンドウの [コマンド] 画面上で help コマンドを実行して表示される情報に基づいています。これらのコマンドの詳細な使用方法については、『IDE ガイド』をご覧ください。

start [引数]	プログラムの実行をデバッグモードで開始。 メニュー：[デバッグの開始]、ボタン：[開始]
run [引数]	プログラムを通常モード実行。 メニュー：[ツール]→[実行(R)]、ボタン：[実行]
step	関数内にステップイン。 メニュー：[シングルステップ]、ボタン：[シングルステップ]
next	関数をステップオーバー。 メニュー：[ステップオーバー]、ボタン：[次へ]
cont	ブレークポイントまで実行。 メニュー：[続行]、ボタン：[続行]
up	呼び出した関数のスタックにアクセス可能にする。 メニュー：[呼び出した関数の中断点に戻る]、ボタン：[上へ]
down	呼び出された関数のスタックにアクセス可能にする。

stack	メニュー：[呼び出された関数の中断点に戻る]、ボタン：[下へ] すべてのスタック内のスタック名を表示
locals	現在のスコープ内の変数と値を表示
variables	すべてのスタックの変数と値を表示
watch 式	ウォッチリストに式を追加
remove 式	ウォッチリストから式を削除
remove	ウォッチリストからすべての式を削除
stopat ファイル名 # [条件]	ソースファイルの#行目に新規のブレークポイントを設定。 メニュー：[ブレークポイントを現在行に設定]、ボタン：[ブレークポイントを設定]。ただしいずれも条件設定は不可
stopin 関数名 [条件]	関数に新規のブレークポイントを設定
stopvar 変数名 [条件]	制御変数に対して新規のブレークポイントを設定
clearline ファイル名 #	ソースファイルの#行目に設定されたブレークポイントを解除。 メニュー：[現在行のブレークポイントを解除]、ボタン：[ブレークポイントを解除]
clearfunc 関数名	関数に対するブレークポイントを解除
clearvar 変数名	変数に対するブレークポイントを解除
clear	すべてのブレークポイントを解除
help	デバッグコマンドのヘルプ情報を表示
assign 変数=式	変数に値を割り当てる
call <i>func()</i>	関数を呼び出す
print 式	式の値を表示
式	式の値を表示
abort	デバッグを中断

3. ライブラリ関数の互換性

3.1 概要

ここでは、Ultra-C Pro で使用されている各ライブラリ関数に対して、Ch Professional との相違点について説明します。

「ライブラリ関数」

ここでいう「ライブラリ関数」とは、Ultra-C Pro の『ライブラリ・リファレンス』で用いられている用語で、標準 C ライブラリの関数を意味し、この章題にある一般的なライブラリ関数とは意味が異なります。Ch Professional は標準 C の機能をすべてサポートしていますので、すべての「ライブラリ関数」に対して完全に互換性があります。

コンソール I/O 関数

Ch Professional は、Ultra-C Pro で使用されているコンソール I/O 関数をすべてサポートしています。

ディレクトリ操作関数

Ultra-C Pro 独自の関数であり、互換性はありません。しかしながら、Ch Professional では MS-DOS シェル、Unix のいくつかのシェルのコマンドをプログラム内でも使用できるため、これらの関数の機能をプログラムの書き換えによって代替することは可能です。

グラフィック関数

Ultra-C Pro 独自の関数であり、互換性はありません。そもそも、Ultra-C Pro と Ch Professional とはグラフィックデータの扱いに関して、以下のような根本的な相違点が存在しています。

出力先：

- Ultra-C Pro ではグラフィックの出力先として、独自の[グラフィック] ウィンドウを使用します。グラフィック出力のサイズはハードウェア画面のドット（ピクセル）単位に固定されます。
- Ch Professional ではグラフィックのプロット出力先として、オープンソースの Gnuplot を使用します。Gnuplot により、二次元だけでなく、三次元のプロット出力も容易に実現されます。またグラフィック出力の倍率、縦横比も自由に変更可能です。さらに Gnuplot は独立したアプリケーションとして独自のコマンドをもち、これらを用いて描画や画面クリアなどを手動で行うことができます。

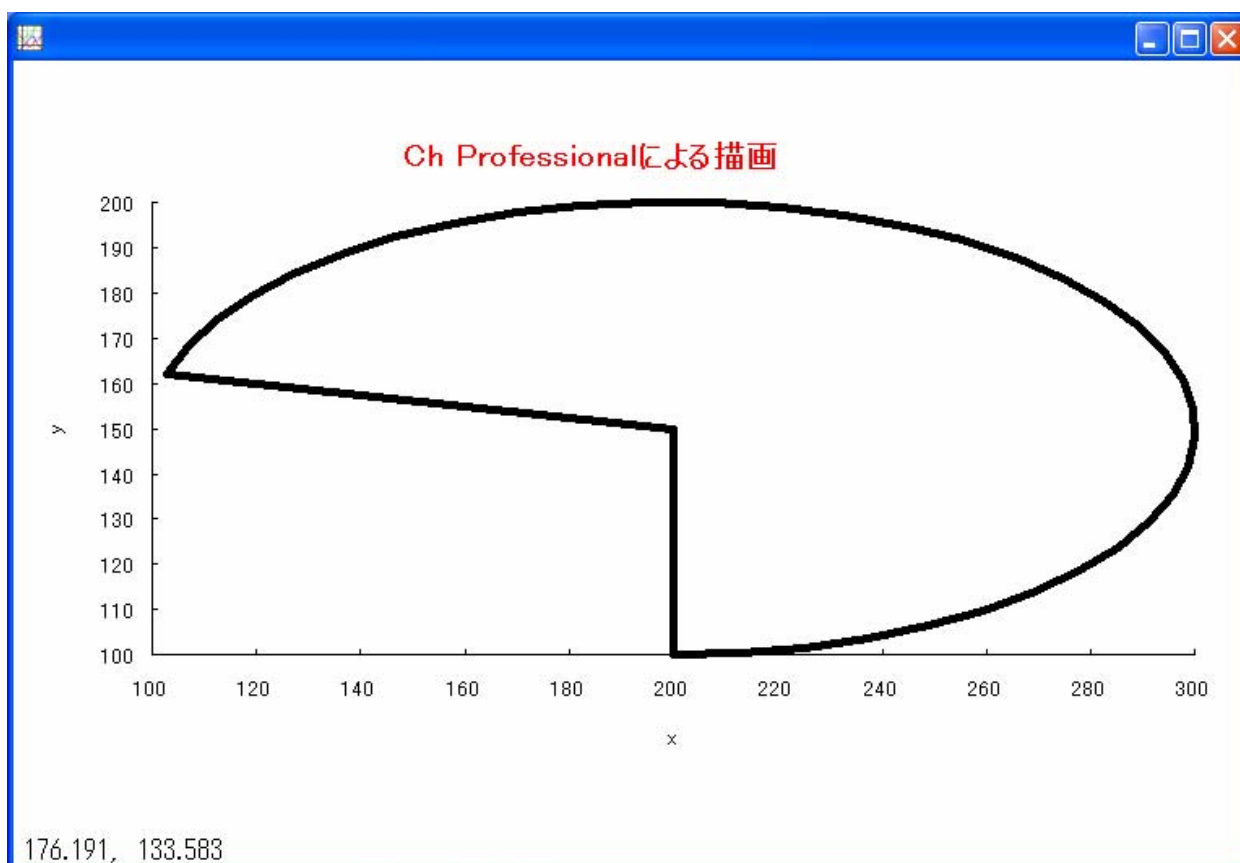
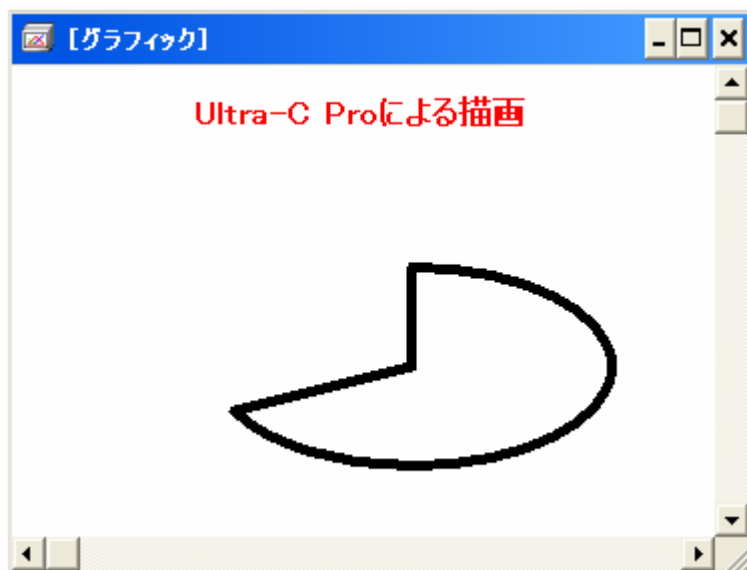
表示座標系について：

- Ultra-C Pro ではグラフィック画面の二次元 xy 座標系として、PC 画面をもとにした、左上隅を原点 (0,0) とした座標系を用います。y 軸は下方向に増加します。
- Ch Professional では、通常数学で用いている xy 座標系にグラフィックデータを出力します。すなわち、y 軸は上方向に増加します。Ultra-C Pro との大きな違いは、y 軸の増加方向が逆であることと、原点 (0,0) を任意の位置に設定可能であることです。

下図は、これらの座標系の違いを示したものです。ここでは Ultra-C Pro のグラフィック関数 (Ch Professional ではその互換プログラム)

```
Pie(100, 100, 300, 200, 80, 180, 200, 80)
```

を用いた、それぞれのアプリケーションの出力結果が示されています。



これらの相違点を踏まえた上で、Ultra-C Pro のいくつかのグラフィック関数を制限付きで Ch Professional 用書き換えることは可能です。ただし、PC のハードウェアや特定のバージョンの Windows の機能に大きく依存した関数に対しては、代替機能の実現は不可能です。このような機能の例としては、以下のものが挙げられます。

- 画像ファイルの読み込みとそれへの書き出し

- RGB データの扱い
- 動的な画像データ（アニメーションのような）の出力
- 破線、一点鎖線など、描画線種の指定
- 背景色の指定。ただし Gnuplot に対して手動で指定することは可能です
- 現在の描画点位置の読み取り。ただし出力結果の Gnuplot 上でのマウスポインタの位置はリアルタイムに表示されます
- Wave ファイルの再生
- ビープ音の発生
- 文字列の色やフォントの指定

実は Ch Professional でも、Win32 API をダイレクトに呼び出し、Gnuplot ではなくて独自の出力用ウィンドウに描画することにより、Ultra-C Pro の各グラフィック関数とほぼ完全に同じ機能を実現することは可能ですが、これでは Ch Professional の本来の機能を使用することにはならず、“互換性”とは全く別の意味になります。

I/O ポート関数

Ultra-C Pro 独自の関数であり、互換性はありません。そもそも Windows NT4 以降の Windows オペレーティングシステムでは、I/O ポートに通常的手段で直接アクセスすることができないようにされています。これらの関数を Ch Professional で代替することはできません。

3.2 サンプルプログラムの互換性

Ultra-C Pro の SAMPLES サブフォルダには、数多くのサンプルプログラムが提供されています。これらは Ch Professional 上への移植性に応じて、以下のカテゴリに分類されます。

(1) ソースコードを一切変更せずに、そのまま動作可能なプログラム

すべての標準ライブラリ関数のサンプルプログラム：

ABS. C	ACOS. C	ARCHK. C	ASCTIME. C	ASIN. C
ASSERT. C	ATAN. C	ATAN2. C	ATEXIT. C	ATOF. C
atoi. C	ATOL. C	CALLOC. C	CEIL. C	CLK. C
CLK_W. C	COS. C	COSH. C	CTIME. C	DIFFTIME. C
DIV. C	EXP. C	FABS. C	FGETPOS. C	FGETS. C
FLOOR. C	FMOD. C	FPUTC. C	FPUTS. C	FSETPOS. C
GETENV. C	GETS. C	GMTIME. C	HELLO. C	KUKU. C
LABS. C	LDEXP. C	LDIV. C	LOCALTM. C	LOG. C
LOG10. C	MBLEN. C	MBSTOWCS. C	MBTOWC. C	MEMCHR1. C
MEMCHR2. C	MEMCPY. C	MEMMOVE. C	MEMSET. C	MKTIME. C
MODF. C	PRINTF. C	QSORT. C	SETVBUF. C	SIN. C
SINH. C	SQRT. C	STRCSPN. C	STRPBRK. C	STRSPN. C
STRSTR. C	STRTok. C	STRTOL. C	TANH. C	TIME. C
VA. C	VPRINTF. C			

すべてのコンソール I/O 関数のサンプルプログラム :

`_GETCH.C` `_GETCHE.C` `_PUTCH.C`

(2) Ch Professional 用にソースを書き換えることにより、類似の実行結果を得られるプログラム
一部のグラフィック関数サンプルプログラム :

Ultra-C サンプルファイル名	使用されている関数とその機能	プログラムの変更点等
ARC.C	Arc: 楕円弧描画、など	プロット機能を用いた代替プログラム
CHORD.C	Chord: 弓形を描画	プロット機能を用いた代替プログラム
ELLIPSE.C	Ellipse: 楕円描画	プロット機能を用いた代替プログラム
LINE.C	Line: 2 点間を結ぶ直線を描画、など	プロット機能を用いた代替プログラム (一部)
PIE.C	Pie: 扇形を描画、など	プロット機能を用いた代替プログラム
POLYGON.C	Polygon: 多角形を描画、など	CPlot クラスの polygon 関数を用いた代替プログラム
POLYLIN1.C	Polyline: ポリラインの描画、など	プロット機能を用いた代替プログラム
POLYLIN2.C	Polyline: ポリラインの描画、など	プロット機能を用いた代替プログラム
POLYLINE.C	Polyline: ポリラインの描画、など	プロット機能を用いた代替プログラム
RECT.C	Rectangle: 矩形の描画、など	プロット機能を用いた代替プログラム
RRECT.C	RoundRect: 角の丸い矩形を描画、など	プロット機能を用いた代替プログラム
SETPIXEL.C	SetPixel: 画面上の座標に点を描画、など	CPlot クラスの point 関数を用いた代替プログラム
TEXTOUT.C	TextOut: 座標位置に文字列を描画、など	プロット機能を用いた代替プログラム (一部)

一部のディレクトリ操作プログラム :

Ultra-C サンプルファイル名	使用されている関数とその機能	プログラムの変更点等
_CHDIR.C	_chdir: カレントディレクトリの変更	chdir などで代用
_GETDRV.C	_getdrive: カレントドライブを取得	pwd などで代用
_MKDIR.C	_mkdir: ディレクトリの作成	mkdir などで代用
_RMDIR.C	_rmdir: ディレクトリの削除	rmdir などで代用

(3) Ch Professional では代替機能の実現が不可能なプログラム

一部のグラフィック関数サンプルプログラム：

Ultra-C サンプルファイル名	使用されている関数とその機能	備考
BALL.C	Ellipse: 楕円描画、など	代替不可 (アニメーションは実行不可)
BALL2.C	Ellipse: 楕円描画、など	代替不可 (アニメーションは実行不可)
BALL3.C	Ellipse: 楕円描画、など	代替不可 (アニメーションは実行不可)
CLRSCN.C	ClearScreen: 画面を背景色でクリア	Gnuplot での手動コマンドで設定
DRAWBMP.C	DrawBitmap: ビットマップを取り込んで、指定した位置に描画	代替不可
GETBKMC.C	SetBkMode: 背景モードを設定、など	代替不可
GETCPOS.C	GetCurrentPosition: 現在の位置を取得、など	Gnuplot 上で位置はリアルタイムに表示されるが、変数に取得することは不可
GETCPOS1.C	GetCurrentPosition: 現在の位置を取得、など	Gnuplot 上で位置はリアルタイムに表示されるが、変数に取得することは不可
GETCPOS2.C	GetCurrentPosition: 現在の位置を取得、など	Gnuplot 上で位置はリアルタイムに表示されるが、変数に取得することは不可
GETPIXEL.C	GetPixel: 指定位置の RGB コードを取得、など	代替不可
GETRGBV.C	GetRValue: RGB カラーから R 値を取得、など	代替不可
GETTEXTC.C	GetTextColor: 文字列の RGB コードを返す、など	代替不可
MSGBEEP.C	MessageBeep: ビープ音を再生	代替不可
PLAYSND.C	PlaySound: WAVE ファイルを再生	代替不可
RGB.C	RGB マクロ関数: RGB カラーを指定したパラメータから合成	代替不可
SETBKCLR.C	SetBkColor: 背景色を設定	Gnuplot での手動コマンドで設定
SETBKMOD.C	SetBkMode: 背景モードを設定、など	代替不可
SETFLCLR.C	SetFillColor: これ以降描画する図形の内部の色を設定、など	データセットごとに指定するため、代替不可
SETFLSTY.C	SetFillStyle: これ以降描画する図形の内部のスタイルを設定、など	データセットごとに指定するため、代替不可
SETLNCLR.C	SetLineColor: これ以降描画する図形の線や枠の色を設定、など	データセットごとに指定するため、代替不可

Ultra-C サンプルファイル名	使用されている関数とその機能	備考
SETLNSTY.C	SetLineStyle: これ以降描画する図形の線や枠のスタイル、太さを設定、など	データセットごとに指定するため、代替不可
SETTXTC.C	SetTextColor: 文字列の色を設定、など	代替不可
SETTXTF.C	SetFont: 文字列のフォントの種類、サイズ、スタイルの設定、など	代替不可
UCPRO.C	DrawBitmap: ビットマップを取り込んで、指定した位置に描画、など	代替不可

一部のディレクトリ操作プログラム：

Ultra-C サンプルファイル名	使用されている関数とその機能	備考
_CHDRIVE.C	_chdrive: カレントドライブの変更	代替不可（直接“c:”のようにシェルで指定）
_GETCDWD.C	_getcwd: 指定したドライブのカレントディレクトリを取得	代替不可
_GETCWD.C	_getcwd: 指定したドライブのカレントディレクトリを取得	代替不可

3.3 グラフィックプログラムの変更例

ここでは、Ultra-C Pro の SAMPLES サブフォルダにある楕円弧の描画プログラム Arc.C を例として取り上げます。Arc.C のオリジナルソースコードは以下のとおりです。

```
#include <igraph.h>

void main(void)
{
    SetLineStyle(PS_SOLID, 5);
    SetLineColor(CL_BLACK);
    Arc(100, 100, 300, 200, 80, 180, 200, 80);
}
```

これを Ch Professional 用に書き換えると、たとえば以下のようなソースになります。

```
//#include <igraph.h>
#include <math.h>
```

```

#include <chplot.h> // プロットに必要なクラス定義

#define CL_BLACK 8

class CPlot plot; // CPlot クラスのオブジェクトを定義

/*
  中心点からの変位の x 成分と y 成分から回転角を得る
*/
double getRotationAngle(double xsize, double ysize)
{
    double tempval;

    if (xsize != 0) tempval = atan(ysize / xsize); //  $\pm\pi/2$  でないとき
    else if ( ysize >= 0) return M_PI / 2; //  $\pi/2$  を返す
    else return -M_PI / 2; //  $-\pi/2$  を返す

    if (xsize > 0) return tempval; // 第 4、第 1 象限
    else return tempval + M_PI; // 第 2、第 3 象限
}

/*
  Ch における Arc 関数の代替。
  「グラフィックス画面」ではなく、通常数学で用いられる xy 座標上
  に描画されることに注意
*/
void Arc(int lx, int ly, int rx, int ry, int xs, int ys, int xe, int ye)
{
    int numpoints = 36; // 描画点の数
    array double x[numpoints], y[numpoints]; // 各描画点の座標値
    array double kaku[numpoints]; // 各描画点の中心点からの角度
    double aa, bb, xc, yc;
    double start, end;

    aa = (rx - lx)/2; // 長径
    bb = (ry - ly)/2; // 短径
    xc = (lx + rx)/2; // 中心の x 座標
    yc = (ly + ry)/2; // 中心の y 座標

    start = getRotationAngle(xs - xc, ys - yc); // 描画開始角
    end = getRotationAngle(xe - xc, ye - yc); // 描画終了角

    linspace(kaku, start, end); // 描画角を start から end まで等分割
    x = aa * cos(kaku) + xc; // 描画角を媒介変数とした x の値
    y = bb * sin(kaku) + yc; // 描画角を媒介変数とした y の値
    plot.data2D(x, y); // 配列(x,y)をもった二次元データセットを

```

```

// CPlot クラスのインスタンスに追加
}

void main(void)
{
    //SetLineStyle(PS_SOLID, 5);
    //SetLineColor(CL_BLACK);
    Arc(100, 100, 300, 200, 80, 180, 200, 80);
    plot.plotType(PLOT_PLOTTYPE_LINES, 0, CL_BLACK, 5);
    plot.plotting(); // データセットを表示
}

```

以下、上記のソースコードについて説明していきます。ここでは、Ultra-C Pro でサポートされていない C++ 言語の機能が使用されていますが、C++ 言語仕様そのものについて解説することは本ガイドの主旨ではありませんので、C++ 言語の知識がなくてもおおまかに理解できる程度に説明をとどめます。C++ 言語に関して詳しくお知りになりたい場合は、市販の解説書をお読みください。

(1) 代替関数の呼び出し

main 関数で呼び出す Ultra-C Pro 独自のグラフィック関数

```
Arc(100, 100, 300, 200, 80, 180, 200, 80);
```

は Ch Professional で

```
void Arc(int lx, int ly, int rx, int ry, int xs, int ys, int xe, int ye)
```

のように代替関数として定義しています。また、

```
//SetLineStyle(PS_SOLID, 5);
//SetLineColor(CL_BLACK);
```

の代替機能として、

```
plot.plotType(PLOT_PLOTTYPE_LINES, 0, CL_BLACK, 5);
```

を呼び出し、最後に

```
plot.plotting(); // データセットを表示
```

で、一組のプロットデータ（ここではデータセットと呼ばれています）を Gnuplot 上に表示します。なお、plotType と plotting 関数については、あとで説明します。

(2) クラスとインスタンスの定義

Arc 代替関数を Ch Professional でコーディングするために、「CPlot クラス」による二次元プロット機能を使用します。まず、

```
#include <chplot.h> // プロットに必要なクラス定義
```

でプロットに必要な関数定義のヘッダファイルをインクルードしています。そして

```
class CPlot plot; // CPlot クラスのオブジェクトを定義
```

では、Ch Professional で二次元および三次元のプロットに必要な関数群を定義する CPlot クラス（上記 chplot.h で定義）を定義しています。

ここでクラスとは、C 言語の拡張である C++ 言語の抽象データ型であり、外見上は構造体と似ていますが、構造体とは異なって、そのメンバとして変数だけでなく関数も含むことができます。上記の行で

は CPlot クラスのインスタンス（オブジェクトともいいます）plot を定義し、プログラム内で CPlot クラスを使用できるようにしています。クラスとインスタンスの関係は、鋳型（クラス）とその鋳造品（インスタンス）のようなものであるとお考えください。CPlot クラスのメンバ関数については、『ユーザーズガイド』の「23 章 2次元プロットと3次元プロット」の「23.1.1 プロットのためのデータ」を参照してください。

(3) 描画用データセットの設定

Arc 代替関数での楕円弧の描画は、実際には

```
int numpoints = 36; // 描画点の数
```

で定義された描画点の個数分の xy 座標の組を始点から終点まで直線で補間して行います。これらの座標値（データ点）は、

```
array double x[numpoints], y[numpoints]; // 各描画点の座標値
```

での配列の各要素として定義されます。ここで、キーワード array は、x および y を計算配列として宣言するための型修飾子マクロです。計算配列とは Ch のファーストクラスオブジェクト（『ユーザーズガイド』の「1.5 計算配列」参照）で、配列型データを整数型等のデータと全く同じように扱って計算することができる配列です。たとえば、

```
x = aa * cos(kaku) + xc; // 描画角を媒介変数とした x の値
```

は外見上は 1 行の計算式ですが、実際には以下のように、配列 x と kaku のすべての要素（これらのインデックスを 0, 1, ... 35 とする）に対する 36 行の計算式と同等です。

```
x[0] = aa * cos(kaku[0]) + xc;
```

```
x[1] = aa * cos(kaku[1]) + xc;
```

```
...
```

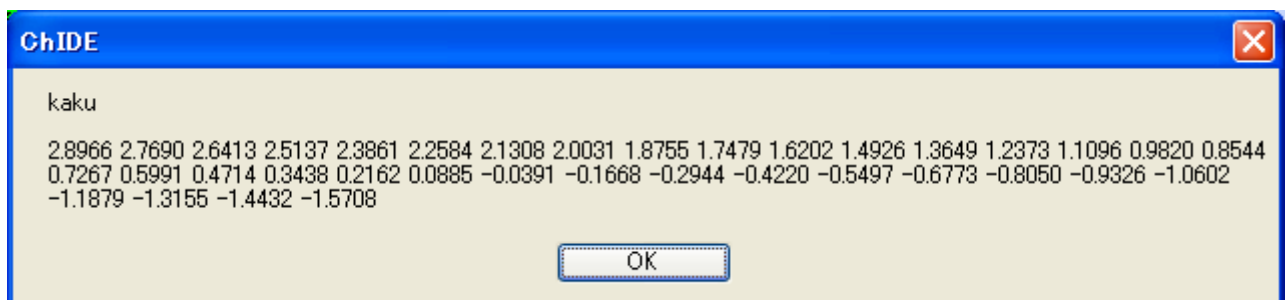
```
x[35] = aa * cos(kaku[35]) + xc;
```

このように計算配列を使用すると、配列の全要素に対する計算処理を 1 行の計算式のみで記述することが可能になり、大変便利です。

このプログラムでは、実際に各要素の計算の前に linspace 関数を

```
linspace(kaku, start, end); // 描画角を start から end まで等分割
```

ように呼び出すことにより、x、y の媒介変数となる描画角 kaku の各要素の値を start（描画開始角）から end（描画終了角）まで numpoints-1（35）等分します（linspace 関数については、英語版の『Reference Guide』を参照してください）。start = 2.8966、end = -1.5708 ですので、実際に kaku の各要素の値はこの行の実行後に以下ようになります（デバッグモードでデバッグウィンドウの「ローカル変数」画面で“kaku”を選択し、これをダブルクリックすると表示されます）。



これにより、

```
x = aa * cos(kaku) + xc; // 描画角を媒介変数とした x の値  
y = bb * sin(kaku) + yc; // 描画角を媒介変数とした y の値
```

は、上記の kaku の各要素の値に対して計算されます。さらに次の行

```
plot.data2D(x, y); // 配列(x,y)をもった二次元データセットを  
// CPlot クラスのインスタンスに追加
```

では、CPlot クラスのメンバ関数 data2D により、プロットの対象となるデータ点のデータセットが plot インスタンスにセットされます。

(4) データセットの描画

このプログラムでは、Arc 関数を実行した時点では二次元のデータ点の組 (x, y) がインスタンスに内部的に追加されたのみです。実際にこれを描画出力するには、

```
plot.plotting(); // データセットを表示
```

のように、CPlot クラスのメンバ関数 plotting を呼び出します。この関数呼び出しは、表示対象のデータセットの設定がすべて完了した時点で行います。本プログラムの場合、

```
//SetLineStyle(PS_SOLID, 5);  
//SetLineColor(CL_BLACK);
```

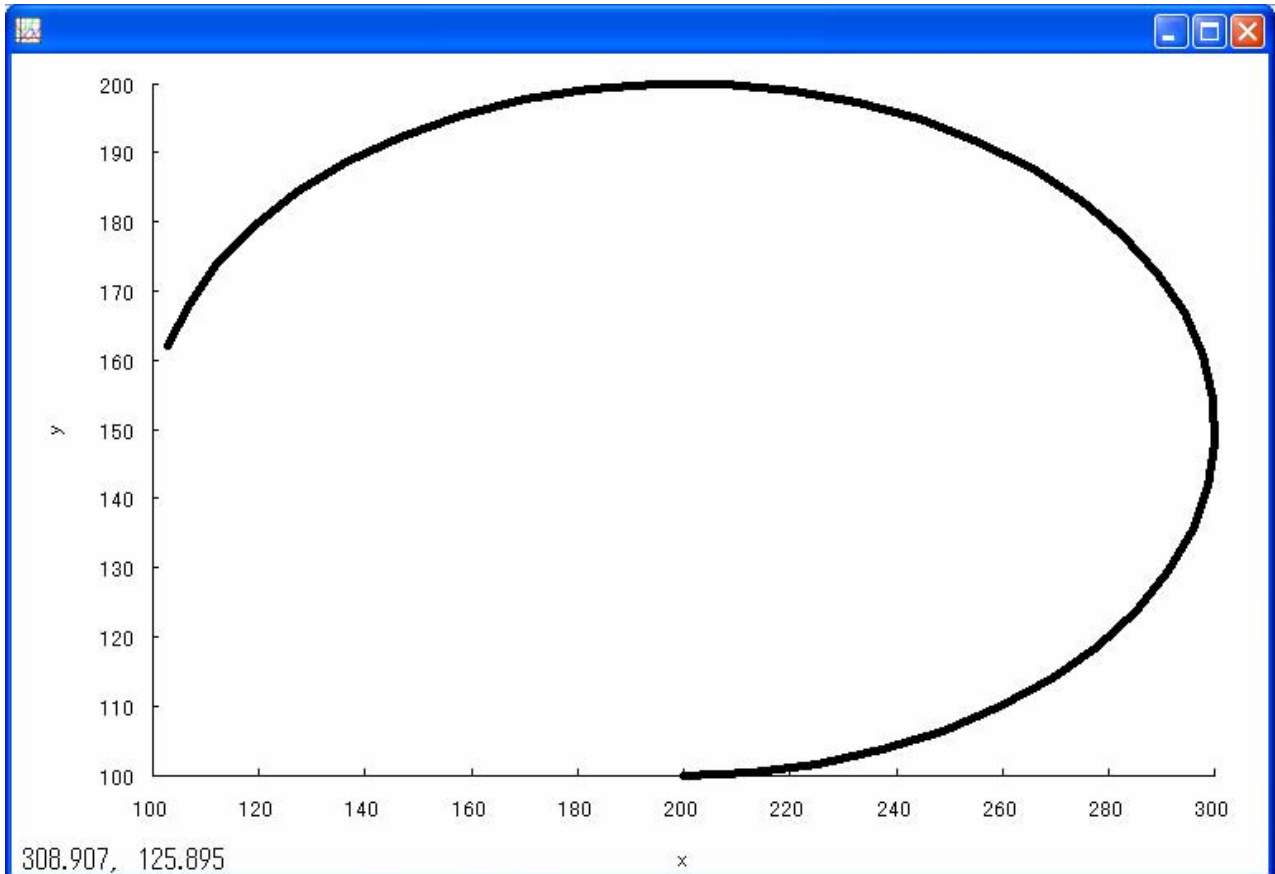
に対応する表示属性の設定も必要となります。これを行うには、

```
plot.plotType(PLOT_PLOTTYPE_LINES, 0, CL_BLACK, 5);
```

のように、CPlot クラスのメンバ関数 plotType により、データセットのプロット属性を設定します。plotType の詳細な仕様については、英語版の『Reference Guide』をご覧ください。

なお、plotType は既存のデータセットに対して**あとから**属性設定を行いますので、本プログラムでは Arc 関数実行後に呼び出します。Ultra-C Pro のように、これから描画するグラフィックに対して**属性を前もって設定することはできません**のでご注意ください。

このプログラムを実行すると、以下のように Gnuplot 上に描画結果が出力されます。



ご注意：

- このガイドですでに述べたように、y 座標の増加方向は Ultra-C Pro の場合とは逆に上方方向になっています。
- 描画ウィンドウ (Gnuplot) 上のマウスポインタの現在位置 (xy 座標) は、左下の数値でリアルタイムに表示されます。
- この描画結果表示中は、プログラムはまだ“実行中”の状態になっています。このプログラムを終了するには、タイトルバー右端の [閉じる] ボタン ('x' マーク) をクリックして、Gnuplot を終了する必要があります。
- 上図のように、デフォルトでは表示されたグラフィックサイズの縦横比は必ずしも 1 対 1 になっていません。この比率を変更するには、描画ウィンドウの外枠をドラッグするか、または Gnuplot のコマンドを用いて設定を行ってください。