VB6 を利用するプロトタイププログラム

Prototype.exe
プログラマ向け説明書

はじめに

コンピュータは、本来、高速の**数値計算**に利用する計算機械です。パソコン（パーソナルコンピュータ）の機能が格段に向上し、大衆化もしてきましたので、コンピュータを計算の道具として利用することが目立たなくなってきました。計算の道具として見るとき、その利用分野は、大きく、科学技術計算と事務計算とに分けます。後者の方は、主に身近なお金の計算に関係しますので、結果として、事務処理を指向したコンピュータの利用が圧倒的に多くなりました。Microsoft 社が Office の名称で発売している一群のソフトウエアが、この実情を表しています。なかでも、表計算ソフトの EXCEL は、豊富な組み込み関数が利用できるように進化してきましたので、科学技術計算でも多く利用されるようになってきました。しかし、一般的に言えば、科学技術計算は専門ごとに固有の問題がありますので、一つの完成プログラムを共同で利用する使い方は多くなりません。商業ベース的に見れば、完成プログラムの販売は魅力に乏しい面があります。したがって、科学技術計算では、プログラミングツールの方に汎用性を求めます。しかし、ベンダーのサポートは不便になってきました。例えば、Fortran コンパイラである MS-Fortran のサポートは、Microsoft 社は打ち切ってしまいました。これに代わるプログラミングツールは幾つかありますが、個人レベルでのソフトウエアの開発と利用には Visual Basic が便利です。

科学技術計算は、専門毎に多様です。何かのプログラムが必要になるとき、出来合いのマーケットソフトウエアを購入できることは多くありません。したがって、従来から、個人ベースでプログラムを開発することが一つの常識でした。そうは言っても、毎回０からプログラミングしていては大変ですので、部品に相当するサブルーチンなどで準備しておきます。パソコンのオペレーティングシステムが Windows の環境になったことで、**オブジェクト指向プログラミング**が要請されるようになりまして。その基本的な骨格プログラムを提供しようと言うのが"Prototype"です。コンピュータの機能が多様化してきましたので、すべてを自前で開発するよりは、パソコンシステムに明るいプログラマの協力が得られる方が望ましい時代になりました。科学技術計算のプログラミングは、事務処理のソフトウエア開発の習慣とは少し違うところがありますので、その違いを、プログラマもユーザも共に納得してもらう必要があります。この冊子の表題が「プログラマ向け説明書」としたのはそのためです。ここでは、やや初心者向けの説明から始めましたが、言わば常識の再確認と穴埋めとを意識しました。この冊子では、コンピュータ関連の専門用語を説明抜きで使うことがありますが、これらは何かの「コンピュータ用語辞典」を参考にして下さい。日本語の辞典も悪くはありませんが、できれば、英語版の Computer Dictionary を参照することを薦めます。

島田　静雄　2007.04.11 版　（全 55 ページ）

内　容　目　次

索　引

索引で参照する番号は、(章番号・節番号・段落の順) を示します。

英字

1. 計算に使う道具

1.1 算術の再認識から

まずは紙と鉛筆が要ること

江戸時代の庶民の階層は「士農工商」に分けられていました。商は低位に扱われていましたが、社会的には重要な役目を担っていました。同じく江戸時代の一般庶民の教養は「読み・書き・算盤（そろばん)」でした。算盤が、商の計算道具として重要であったことを意味しています。あまりにも常識的ですので気が付き難いことですが、一寸した計算でも用紙と筆記用具を使います。計算に使うデータも用紙に書いてあるのが普通です。計算結果を書類として残す場合には、**書式**(form)を整えて清書しなければなりませんので、計算作業とは異なって「書き」のための神経を使います。字が下手であると肩身の狭い思いをします。**簿記**は、bookkeeping の発音を当てた巧みな訳語です。お金に関係する計算の手順と共に、決められた書式にまとめる技術であって、欧米からの学習が始まりです。簿記は、特殊な専門分野と思うでしょうが、その裾野は日常的な庶民レベルのお金の記帳にまで広がっています。コンピュータソフトウエアの EXCEL は、大衆化したプログラムに発展しましたが、その理由は、簿記を指向した機能であることに加えて、計算の煩わしさと清書の苦労を、大幅に解決してくれるからです。

算盤の使い方を常識としたいこと

小学校では、教科の名前として算数と言いますが、算術の方が古い言い方ですし、実践的な**技術**の意味があります。筆算は、紙と鉛筆以外に特別な道具を使わない計算技術を構成しています。知識として覚える数学のような学問ではありません。一般論で言うと、技術は、**道具**と**技法**と**技能**の三つで構成されます。算盤を使う計算を考えると分かるように、道具が算盤、技法が玉の使い方の約束、そして算盤を使いこなすための技能が必要です。算盤使いの上手・下手を表すために、技能を段級のランクで評価することがあります。「指折り数える」方法は、指を道具として使います。上で説明した紙と鉛筆も道具に位置付けます。ただし、**書き方**は別の技術を構成しますので、狭く考えるときには一緒にしませんが、非常に重要です。

算術は正の整数の計算を基本とすること

気が付き難いことですが、算術には、数の概念に負（マイナス）がありません。小学校低学年の算数は、足し算を先に学習し、その知識の応用として引き算を習います。引き算の演習問題の答えは**余り**です。２桁以上の数の引き算も１桁の引き算の応用です。同じ位取りの１桁同士の引き算で引けないとき、上の桁から 10 を借りてくる技法を習います。機械式の計算機では、全体として引けないと、頭に 9999…と並ぶ数が現われます。これが**補数**表現です。電卓ではマイナスの数表示が得られ、補数表現が現われません。小学校の算数では、引けない引き算の問題は、最初から含ませていません。算盤も、正の数だけを表示する道具です。算盤を使う計算術では、赤字になるお金の計算は引けなくなったときですが、このときは、仮に別の桁を使って正負を入れ替えて引き算の計算をします。赤字と黒字とは別枠の数のグループとして扱い、混ぜて使うことはありません。つまり、負になる数の扱いは案外難しいのです。

普通の電卓は余りの計算が直接にはできない

掛け算は、同じ数を何回も繰り返して足し算をすることです。逆に、割り算は同じ数を何回も引き算をすることです。引けなくなった残りが余り（剰余）です。電卓が安価で便利に使えるようになりましたので「小学校で九九を覚える必要がない」とする過激な主張があります。しかし、普通の電卓は、整数計算の「10 割る 3 の答えは、3 余り 1」が直接にはできません。答えとして、3.333…の方が正しいと主張するのは視野の狭い人です。余りの計算は、実践的には良く現われます。例えば、或る寸法を一定ピッチに区切ると、端数の長さが残ります。これが余りです。コンピュータでは剰余計算の関数として、例えば EXCEL には MOD（ラテン語の modulo から）がありますが、知らない人も少なくありません。余りを無視する割り算の計算は、小数以下を切り捨てる計算です。科学技術計算は、数学的に数値を正確に扱わなければなりません。しかし、数値を書類に残す場合には、長い桁数を短く端折り、実質的には小数点位置を便宜的に考えた整数扱いをします。このような扱いの数を**呼び数**と言います。典型的な呼び数は円周率を表すときに扱われています。3, 3.14, 3.1416, などを場面に応じて使い分けます。「円周率を 3 と教えるのは正確な表現ではない」と目くじらを立てる人もいます。数値計算は技術ですので、実践的また習慣的な規則を覚えることも技能勉強の一つです。

1.2　実数計算の工夫

コンピュータも基本的には算術の道具である

コンピュータは、算盤と同じように、基本的に正の整数を計算する道具です。算盤は全体で一続きのレジスタになっていて、適当な区間を別々のレジスタに分割して数を置きます。コンピュータは、8 ビット、16 ビット、32 ビットのような二進数のレジスタが幾つかあって、必要に応じてそれを論理的に繋いで長いビット数のレジスタとして使います。レジスタは正の整数を扱いますので、そのままでは負の数を表すことができません。そのため、負の数を表すビット並びの約束がソフトウエアで決めてあります。整数の掛け算をするとき、5 桁×5 桁の計算結果は、10 桁長さのレジスタがないと正しい結果が得られません。簡単な電卓では、小数点を含む実数計算でも整数計算を基本としていますので、整数部分の桁数が表示桁数を超えるとエラー表示が出ます。整数桁数が大きくなると浮動小数点数の表示になる電卓は、やや高級機です。Windows 系のパソコンでは、アクセサリに電卓がありますが、これはコンピュータ本体の CPU を内部で使っていますので、オーバーフローやアンダーフローのエラーは出ません。

小数を含む計算も整数計算で行っていること

日本のお金の単位系は、古くは円・銭・厘とありました。今は１円が最小単位です。ドルでは 1/100 単位のセントがありますので、数値として扱うときはセントを小数２桁で表記します。しかし、実質的なお金の計算は、全体を通してセント単位の整数として扱い、印刷のときに体裁を整えます。円でもドルでも、利息計算などで小数以下の数を計算に使っても、結果は小数以下を切り捨てる処理が基本です。ここで言いたいことは、数値の扱いが、事務処理と科学技術計算とでは微妙に違うことです。お金の計算は、桁数が多くても、最小単位の数値まで正確さが要求されます。科学技術の計算では、四捨五入・有効数字・概数・精度などの扱いが一定ではありません。学問的な興味から、長い有効数字を持つ実数を扱うこともありますが、実用的には小数点の位置を便宜的に付け変える処理をしながら整数同士の計算が行われます。32 ビットのレジスタで表現する実数の精度は 10 進数で約 7 桁の精度になりますので、特に数学的な精度を問題としなければ、書類に使う数値は５桁程度に揃えます。このときに使う恣意的な整形を**フォーマット処理**と言います。これが数値計算のプログラミングでは手の掛かる部分です。報告書などにまとめるときの書式や数値の扱いを見れば、その担当者の技術レベルが分かります。このことを逆手に取って、コンピュータからの生の桁数の多い数字を並べて、さも正確な計算が行われたように見せかけて信用させることも行われました。

単位系を一緒に考えること

外貨との交換などを考えなければ、日本ではお金の単位は円一種類です。ただし、桁数が多くなると、漢数字の千・万・億などを単位にした表記を使います。一方、科学技術計算で扱う数では、物理的な単位系(unit)を背負った数値を扱います。長さで言えば、mm, cm, m, km のどれを使っているかで、小数点位置の違う数表記になります。計算処理では実質の数部分を使い、小数点位置を後で調整します。コンピュータで実数を扱う内部表現は、二進数表現にした浮動小数点の数(floating point number)です。これは、数の実体を表す**仮数部**と、位取りを示す**指数部**の二つで構成しますので、加減乗除の計算が複雑になり、整数計算よりも演算時間が余計にかかります。計算結果をレポートとして利用するときには、さらに単位系の調整をします。機械式の計算機を使って数値計算をしていた頃は、小数点位置を無視した仮数部を整数扱いで計算しておいて、単位系などを勘案して小数点位置を調整した表記法を使いました。書類として計算書にまとめるときは、記号表記の計算式を書いて、その通りに数値を代入した式を示して計算結果を表記します。そうすると、単位系を調整する計算が必要になりますので、数値だけを見ると、単純な数値計算結果と小数点位置の違う表記になることがあります。

数のグループとして扱うとき

科学技術計算の結果は、独立した数で利用するときと、数のグループを数表やグラフにして全体としての傾向を見る利用法とがあります。計算結果を示すとき、数の大小幅の大きい数を扱う方法が二通りあります。一つは、小数点位置を固定して一定長さの桁数で揃える方法であって、考え方は整数と同じです。桁の長さは扱う数の絶対値の最大を考えて決めますので、その最小桁位置に満たない数は０とします。この方法が普通です。もう一つは、浮動小数点数の仮数部を重視する表現です。数の大小幅が大きい場合の表し方に向きます。しかし、グラフに表すときには困ることもありますので、対数尺度のグラフで表す表現も考えます。数を文字並びで表記するだけでなく、グラフィックス表示が重要です。

1.3 印刷に使う道具

コンピュータシステムは道具の集合である

コンピュータ本体を含めた全体は、種々の道具とプログラムとから構成されますのでコンピュータシステムと言います。道具に代える言葉として、装置、オブジェクト、デバイスなどの言い方があります。ソフトウエアまたはプログラムは技法に当たります。使いこなすための技能はユーザに要求されますが、それを**ユーザインタフェース**と括るようになりました。技能には、キーボードやマウスの使い方があり、英文タイピングの技能は、必須です。電卓は、最も単純化したコンピュータであって電子算盤の異名を持つように大衆化しました。その技法は、最初から組み込まれたプログラムです。キーを打つことは技能です。速く打てることが高度な技能であって、銀行のような計算事務の多い職場では、算盤技能に代わる必須の技能になりました。身分制度の厳しい社会では、技能が要求される職種を低位に見る嫌いがあります。パソコンは、ユーザに特別な技能を要求しないように進化してきました。自動化の研究は技術の究極の理想でしょうが、実は、コンピュータシステムを便利にし過ぎて技能不要に進化すると、多くの不都合も起こるようになりました。設計プログラムなどは簡単になり過ぎて、少し操作を覚えるだけで、小学生でも専門家並の成果が得られます。これが技術の空洞化に繋がります。安全対策と技術伝承の方法には、意図的に不便で複雑なままにしておく、と言う皮肉な方法も使います。

印刷装置が無いコンピュータは安価であること

手を使う算盤や電卓の取り扱いが幾ら早くても、数を入力し、計算結果を紙に写す出力作業に、かなりの時間を取られます。この作業のときに、入力間違い、写し間違いが良く起こります。間違えないようにする慎重さも大切ですが、間違いを発見する方法も工夫されます。一つの方法は、入力・出力ともにデータをチェック用印刷で残して検査に使うことです。計算機械に印刷機構を付けるのが最も単純な方法であって、商店で使うレジスタがそうです。企業の実務では、コンピュータシステムに印刷装置が必須です。コンピュータの印刷装置には、以前、**電動タイプライタ**を使いました。機械産業の底が浅かった時代には輸入品に頼りましたので、高い買い物につきました。このこともあって、コンピュータメーカーの出自には、電報やテレックスの送受信に使う**コンソールタイプライタ**を製造していた企業が多いのです。電卓は機械部品を殆ど必要としませんので、多くの企業が競合し、価格が劇的に安くなりました。大型計算機（メインフレーム）を使う計算センタは、計算機本体よりも大型の高速印刷機（**ラインプリンタ**）を置く場所の方が広い面積を占めていました。

プリンタが無いと仕事にならないこと

科学技術に関係する計算でも、EXCEL を利用することが少なくありません。しかし、一般的に言えば、簿記のような定型化した標準書式の提案ができません。科学技術計算のプログラミングでは、問題ごとに計算結果を表す書式を考えます。書式を含め、レポートなどの文書にまとめる技術の重要さは軽視されています。手回し計算機・そろばん・電卓などは、計算の道具ですが印刷機能を持っていませんので、計算結果を紙に書き写す作業に手間が掛かります。そこで、印刷機能を持った計算機が要望されるのです。商店でごく普通に使うキャッシュレジスタは、最も単純な印刷機能を持つ計算機です。科学技術計算にも印刷機能のある計算機の要望は強かったのですが、高速プリンタを持つコンピュータが実用になる前には、科学技術計算に利用できる印刷計算機がありませんでした。マイコンと接続して利用できる安価な**ドットプリンタ**(dot matrix printer)の出現は、企業だけでなく、個人レベルでのコンピュータ利用を加速させることに繋がりました。

印刷仕上がりを良くしたい要望が強かったこと

事務処理にコンピュータを使う場合には、仕上がりのきれいな印刷にすることに神経を使います。プリンタの性能が低かった時代には、プリンタ出力を原稿にして、改めて書式を整えた印刷物に作り直すことも少なくありませんでした。科学技術計算は、計算結果の方に注意を向ける傾向が強いので、印刷の質や体裁にはあまりこだわりませんでした。論文として体裁を整えた印刷をするとき、原稿の編集は、印刷屋さんがしてくれたからです。しかし、**DTP**(desk top publishing)が普及して、自前で最終的な完成原稿を作成する時代になって、改めて印刷物作成にからんだ、細かなプログラミング技法を覚えなければならなくなりました。これには二つの問題があります。一つは、プリンタなどのハードウエア環境（物）を理解すること、そしてもう一つが入出力文（扱い）を理解することです。この、物と扱いの二つを繋ぐ概念が、ユーザインタフェースです。

1.4 図を描く道具

グラフに描く表現は絵と違うこと

言葉だけで説明するよりも、図を同時に使うと、ずっと詳しい情報を相手に伝えることができます。このときに使う図は、芸術的な絵画とは質が違っていて、明確な情報伝達の目的を持たせます。工業設計に使う図は、不必要な絵画的要素を省き、形と寸法の必要充分の情報を含ませるような製図法で描きます。文字は、言葉を図形にして表す方法ですので、図面に文字も記入しますが、文書とは使い方が違います。やや絵画的な要素を持たせて説明に使う図を、総称でイラスト(illustration)と言います。写真も、文書の中で使うときはイラスト扱いです。これらは、言葉では説明でき難い感覚的な性質を伝える目的があります。しかし、漫画は大流行ですが、子供じみたムード的な挿絵は使うべきではありません。科学技術計算では、数値の並びをグラフに描くイラストを使います。科学的な実験観察では、現象をレコーダでグラフ化してから解析に入ります。このレコーダが**線図**(line drawing)を描かせる製図用道具の原点です。

平面図形を描くのがプロッタ

測定データをグラフ化するレコーダの横軸(x 軸)は、普通、時間軸です。円のような幾何学的な図を作図するには、ペンを二方向(x,y)に動かすメカニズムが必要です。これを X-Y レコーダと言います。電気信号を生で使うアナログ方式のレコーダは精度に問題がありますので、デジタル式のレコーダが開発されました。これがプロッタです。最初の利用は、曲線の多い大寸法の地図を正確に描く目的の利用でしたが、これを自動製図機械に応用することで普及しました。初期のプロッタは、ペンを動かすメカニズムでしたが、現在では大寸法のレーザプリンタやインクジェットプリンタを使うようになりましたので、プロッタとプリンタの区別がなくなりました。しかし、作図の原理やアルゴリズムを理解するときは、ペンを動かすプロッタを制御することを考えます。ペンを動かして描く図を線図というのに対して、イラストなどは**濃淡図**を主に使います。英語では painting です。作図原理は、細かな点（ドット）の集合で構成します。１ドット単位が小さく、その集合密度が大きければ精細な図が描けます。ドットの集合で線を構成するように作図しても、ギザギザが気にならないようになりました。

用器画は道具を使う作図法

定規・コンパス・分度器などの道具を補助的に使って描く幾何学的な図を**用器画**と言い、これに対するのを**自在画**と言います。製図は用器画です。上手な図を描くには、道具だけでなく技能も磨かなければなりませんので、それを支援する技術として自動製図(CAD: computer aided drafting)が研究されました。手を使う作業の質も変りますし、コンピュータを利用する技能が要求されるようになりました。線図は、工業製図よりも、印刷物のイラストに使う方が一般的です。こちらの方は、もっぱら印刷の仕上がりに関係する立場で作図技法が扱われます。細過ぎても、また、太過ぎても良くありません。例えば、大きな寸法の図面を写真技法で縮小すると線が細くなり過ぎますので、書き直すことになります。図の大きさを変えて用紙寸法に合わせた線の太さで描かせることは、自動製図では重要な機能です。図形そのものではなく、図形を描くの幾何学的データをファイルに保存しておけば、**再描画**に対応することができます。このファイルを**メタファイル**と言います。コンピュータのモニタ画面を画板に見立て、マウスなどを使って自在画風に作図するのでは再描画ができません。したがって、作図は総てコンピュータの命令語の集合で書かせるようにプログラミングをします。

A4 版の寸法までの大きさの図が実用的であること

構造物の設計図などは大きな寸法の用紙を使いますが、書類に使う図は最大で A4 版の寸法が実用的です。A4 版用紙を使うプリンタが普及しましたので、特に理由がなければ A4 版以上の書類を作成するのは避けます。コンピュータのモニタも、14 型は横位置の A4 版の寸法です。約 30cm の明視の距離で見るときに、全体が見易い視野に入ります。デモンストレーションに使う場合を別として、モニタ上で確認した文字や図を、そのまま、ほぼ同寸法でプリントすることを **WYSIWIG**(what you see is what you get)と言います。実務でコンピュータを利用するときには、成果を何らかのハードコピーに作成しますので、プリント成果品の絶対寸法を意識して作業をします。ワードプロセッサは、用紙と文字フォントの選択には注意深い工夫がされています。モニタの画面上だけでコンピュータグラフィックスを楽しむ分には、寸法のことをあまり意識しなくても済みます。数値計算の結果をグラフに作図する場合には、意図した寸法で見易く仕上がるようにするため、座標変換などの幾何学的処理に手が掛かります。

1.5　標準入出力環境の構想

文字の表示が基本であること

DOS の環境に設定されたパソコンでのモニタ画面は、文字の表示が主目的です。そのため、このモニタをテキストディスプレイと言うこともあります。その標準的な文字並びは、半角 80 文字×25 行でした。キーボードからの文字入力は、すぐにモニタにエコー表示されます。25 行を使い切ると、行が繰り上がって上の行がなくなり、新しい行が最下行に書き込まれます。消された行を見ることはできません。プログラム文でモニタに書き出す文字並びも同じ扱いですので、行数の多くなるリストをモニタで確認する目的には向きません。そのため、リストを後で確認するため、平行してプリンタを使いました。そうすると、文字の書き出し文が２種類必要です。モニタ用とプリンタ用です。DOS 時代のパソコン用 Basic 言語では、PRINT 文と LPRINT 文とがありました。コンソールタイプライタやドットインパクトプリンタは行単位でプリントしますので、プログラムの実行時にプリンタに書き出して、モニタのようにも使えました。レーザープリンタやインクジェットプリンタは用紙単位で印刷しますので、一旦データをバッファーに貯めて、まとめてプリンタに書き出します。そうであるなら、出力文は最初からデータをファイルに書き出しておく方が便利です。

文字とグラフィックスは表示原理が異なること

科学技術計算では図の利用はほぼ必須と言えます。ムード的なイラストや漫画的な図をではなく、説明目的を持たせます。コンピュータを利用して図の作成をするには、始めのころ、プリンタとは全く別原理の、ペンを動かすプロッタを使いました。プロッタは色を使う作図には向きません。カラーテレビのように、モニタ上でグラフィックスを扱うためには、解像度の高いカラーグラフィックスモニタが必要でした。これをグラフィックスディスプレイとも言いました。その解像度は、当初、640×480 ピクセルでした。現在ではずっと解像度の高いモニタを使うことができます。グラフィックス表示と文字表示とは、ソフトウエア的には作図原理が異なります。DOS の環境では、文字専用で使う場合とグラフィックスに使用するときは、ソフトウエア的に画面を切り換えました。モニタ上のグラフィックス画像は、テレビの画面と同じように単純な観察目的に使いますので、グラフィックスの成果をレポートに使いたいときは、写真に取って貼付けました。しかし、モニタの解像度は写真の解像度よりも低いので、質のよい写真は得られません。高品質のレーザープリンタやインクジェットプリンタが利用できるようになりましたので、ペン書きのプロッタは無くなりましたが、作図の命令文は、プロッタの動作原理も引継いでいます。プロッタは大きな寸法の図面を扱いますが、モニタは相対的に小さな画面です。したがって、モニタ上の図をそのままプリンタに描きだすのでは質のよいハードコピーが得られません。そのため、作図用のデータをファイルに書き出しておいて、グラフィックス装置に合わせて作図させることも行われます。このファイルをグラフィックス用の**メタファイル**と言い、CAD で良く使う方法です。

三つのウインドウを準備すること

Windows のパソコン環境は、DOS の画面単位に相当する枠を複数表示することができます。この枠を通称でウインドウと言うことから、複数形の Windows の登録商品名が使われるようになりました。プログラミングの用語では、この枠を**フォーム**(form)と言い換えます。モニタ全体はデスクトップ(desktop)です。Windows では、モニタ上に文字やグラフィックスを表示する方法が多様に選択できるようになりましたが、目的に合わせて複数のフォームに分けて扱います。プログラム"Prototype"では、DOS の画面単位を三つ使うような、三つのフォームを考えることにしました。第一は、標準的なモニタ画面、第二はプリンタに書き出すリストのモニタ用、第三はグラフィックス用のモニタです。第一のモニタ画面は、DOS のコンソールをモデル化しますが、主としてユーザ向けの表示専用で設計します。第二は、プリンタに出力させる原稿のバッファーの目的を持たせます。この画面は、テキストエディタの機能を持たせ、スクロール機能が使えます。第三のグラフィックス用のフォームも、報告書などに使うハードコピーを作成する原稿をモニタする目的に持たせます。通常は最大で A4 版の用紙に納まる大きさで図が得られれば良いので、特にメタファイルを使うまでもなく、そのままプリンタに書き出すか、ビットマップファイルで保存するように設計します。この三つのフォーム（通称でのウインドウ）を総合的に管理するため、一つの親ウインドウを考え、三つのフォームを子ウインドウ扱いにします。これらは、データ入出力と関係するハードウエアの性格がありますので、これを擬似的な装置、つまりオブジェクトの機能設計をします。DOS の環境ではモニタが一つでしたが、Windows の環境では複数のフォームに名前をつけて識別し、その名前で修飾した命令文を使います。

1.6　標準のフォームデザイン

フォームはオブジェクト指向プログラミングで設計する

Windows の環境で表示する幾つかのウインドウは、プログラミング上の用語ではフォーム(Form)です。フォームは、DOS のモニタ画面に相当しますが、そこに擬似的な装置や部品の図柄をまとめます。これらが総称でオブジェクト（物の意味）です。フォームは、そのままでも文字や図形の表示に使うことができますが、文字の扱い専用には編集機能を持ったテキストボックスを載せます。これらオブジェクトの幾何学的な形状寸法など（**プロパティ**と言います）の定義と、その装置を制御するプログラム命令（**メソッド**と言います）が決められています。VB6 のプログラミングの最初は、フォームの設計から始めます。これが通称で言う**オブジェクト指向プログラミング**です。

"Prototype"で採用したフォームのデザイン

プログラム"Prototype"の実行画面の一例を図 1.1 に示します。この全体枠は親ウインドウになっていて、その中に三つの子ウインドウ(子フォーム)を配置します。これが **MDI** (Multi Document Interface)方式です。大枠を**親フォーム**(MDIFormConsole)とし、その構造は、上から**タイトルバー**、**メニューバー**、その下が**クライアント領域**、最下段に**ステータスバー**を付けました。ステータスバーは、幾つかのパネルに分けてあって、それぞれがファンクションキーのように機能させる設計にしました。三つの**子フォーム**の呼び名は、**コンソール**(FormConsole)・**テキスト出力画面**(FormLprint)・**グラフィックス画面**(FormGraphic)としました。子フォームは親フォームのクライアント領域の中で位置と寸法を自由に変更できます。子フォームは、タイトルバーとクライアント領域だけの構成です。VB6 の仕様では、子フォームに関連するメニューは、設計時には子フォームに付属していますが、プログラムの実行時には、アクティブになった子フォームのメニューが親フォームに表示されます。なお、ツールバーは使いません。これはメニュー項目をアイコン化して表示する目的で使いますが、アイコンを準備しなければならず、また、メニューと機能的に重複しますので、意図的に省きました。

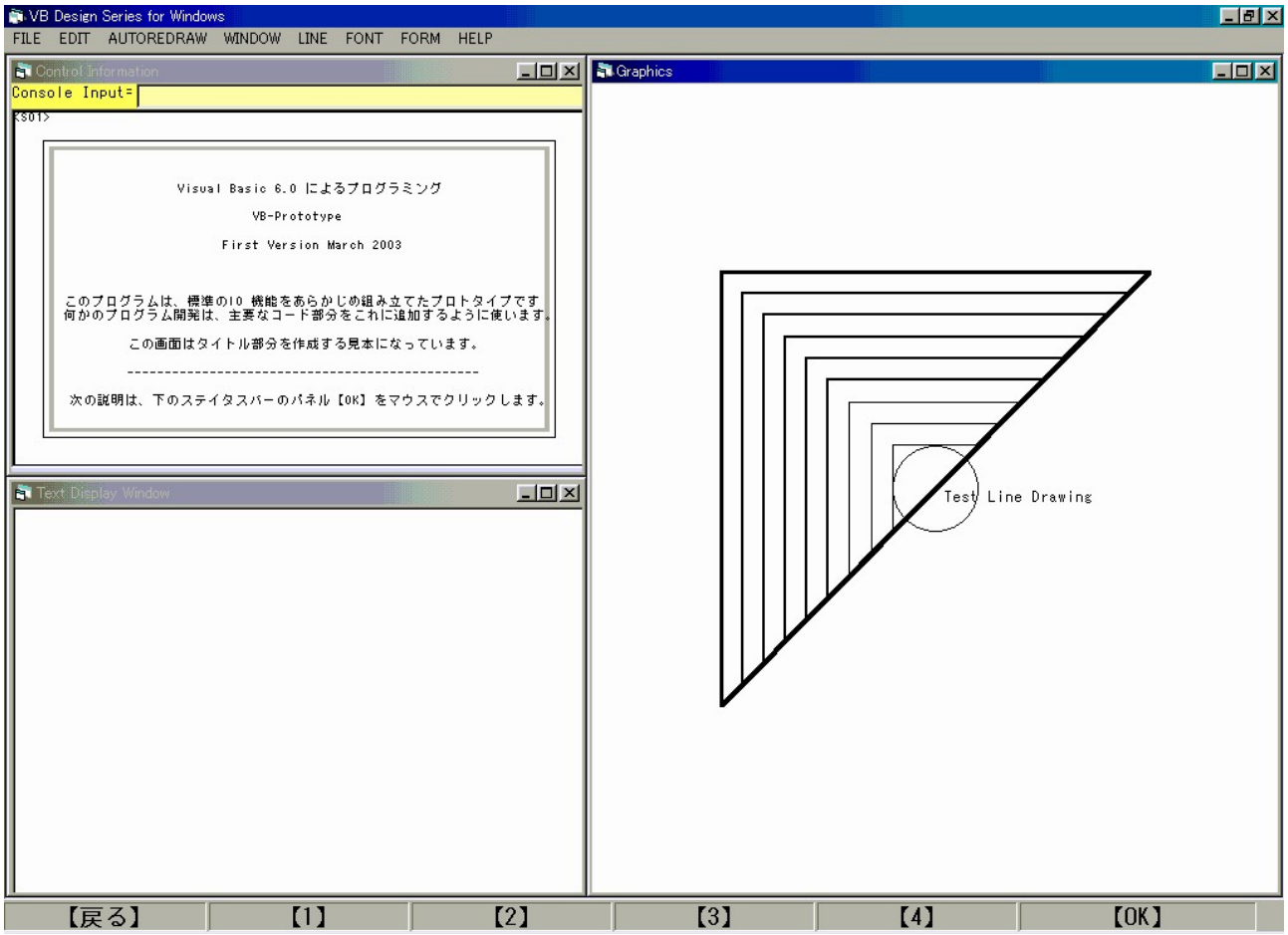

図 1.1: 三つの子ウインドウは、左上が FormConsole、その下に FormLprint、右半分は FormGrahic です。FormGrahic がアクティブになっていますので、メニューはグラフィックス用の表示です。

1.7　基本的なユーザインタフェース

タイピングの技能が必須であること

コンピュータは原理的に電気電子装置ですので、その制御は基本的に電気的なスイッチを使います。キーボードはスイッチの集合体です。英文タイプライタとしての使い方が標準ですので、コンピュータを使うにはタイピングの技能が必須です。慣れてくれば 100 ものキーの種類があっても、キーの位置を探すことは気になりません。人との対話と同じように、決められた文字並び（コマンド）を入力すると、コンピュータが何かの処理をするようにした方式が CUI(Character User Interface)です。しかし、タイピングに慣れていないユーザも少なくありませんので、あらかじめ機能を割り当てたキーを押す方式も応用されます。その目的に使うキーをファンクションキーと言い、キーボードの最上段に F1～F12 のように並んでいます。GUI(Graphical User Interface)の操作方法を簡単に要約すれば、ファンクションキーに相当する図柄がモニタ上にあって、それをマウスで選択してクリックすることに代えた方法です。その図柄は、アイコンの場合もあり、文字並びを書いたボタンなど、種々のデザインが工夫されています。メニューバーを使うのが標準的な方法です。GUI の便利さを強調するために、CUI を時代遅れのインタフェースであると低位に見る傾向もありますが、両方の長所を生かす工夫が必要です。テキストエディタやワードプロセッサは、キーボードが無ければ仕事にならないからです。

処理の流れを制御する方法

科学技術計算にコンピュータを利用するときの一つの特徴は、条件を与えて答えが要求される処理が多いことです。その答えが出たところまでを一つの区切りとし、その答えを含めた新しい条件で、別の一区切りの計算を繋ぎます。プログラミング技法では、区切り単位をサブルーチンや関数にまとめます。電気電子回路との類似で、この処理の流れを回路図のように描いた図を**流れ図**(flow chart)と言います。条件を最初に入力すれば最終的な答えが出る計算では、ユーザの作業は、最初の条件入力と結果の利用方法だけです。計算処理の途中経過を見たいときは、適当な区切り箇所で処理を中断して、モニタで観察するようにプログラミングをします。中間の結果を見て、幾つかの異なった処理に分岐させるか、条件を入れ替えて再計算させたいことがあります。このときにユーザの介入する方式を、コンピュータと**対話**すると捉え、英語では形容詞の **interactive** で表します。具体的にプログラミングをするときは、モニタ画面に何かの案内メッセージを表示させ、幾つかのメニュー選択肢のどれを選ぶか、キーボードからの入力を受けつけるか、などの設計をします。科学技術計算のプログラミングは、このようなユーザインタフェースが工夫されます。この方式を**ウイザード**(wizard)と呼んでいます。意味は魔法使いであって、コンピュータ側がウイザードの役目を持ってユーザを賢く指導してくれることを指します。

"Prototype"はウイザード方式で設計したこと

全く予備知識なしで何かのアプリケーションを使うとして、その立ち上げ画面を見て、どのように操作で始めればよいかに迷うことは少なくありません。別冊になったマニュアルを見るか、readme のファイルがあれば開くか、HELP を参照するか、などの方法を試すのですが、そのどれも無い場合があります。あったとしても、当面の作業画面から脱線します。そこで、現在の画面で、何をすればよいかの案内表示があると親切です。これが要するにウイザード方式です。メニューの使い方は、幾つかの選択肢がランダムに選べる処理方法ですので、処理の流れから見れば、本流から逸れた、言わば道草的な処理です。メニューでの処理が済めば、元の本流に戻ります。そのため、メニュー処理に重要な役目を持たせて、必ずそれを実行させるようにするのではなく、選択肢の一つをあらかじめデフォルトの設定にしておきます。幾つかある選択肢のどれかを必ず選ばなければ、次の処理に移れないような方法の代表的なものは、メッセージボックスです。例えば、Yes, No, Cancel と表示されたボタンを使います。"Prototype"では、親ウインドウの下段にあるステータスバーに幾つかの**パネル**を割り当て、それをボタンとして使います。基本的には、【戻る】【1】【2】【3】【4】【OK】と表示してあります。パネルを選択したときに何が行われるかの案内をコンソール画面に表示します。図 1.1 は"Prototype"立ち上げ画面です。左上がコンソールです。この画面は、文字だけでなく、簡単なグラフィックス表示も使うことができます。例図では、枠を線図で描いてあります。プログラムは幾つかの段階（ステータス）に分けてあって、その段階を区別する番号をコンソールの左上に、<S01>のように表示します。原則としてパネル【OK】をマウスでクリックすると次のステータスに移り、コンソールの表示も切り換えます。

1.8 Windows の標準機能の利用

タイピングの技能は常識であるとしたこと

コンピュータは数値計算を目的として開発されたのですが、案外なことに、実際作業時間の殆どは文字処理に使われています。プログラム文の編集作業はテキストエディタが主役ですし、レポートなどで体裁のよい書類にするにはワードプロセッサを使います。そのこともあって、コンピュータを使うときに要求される技能の大部分は、英文タイピストが覚える技能の性格があります。まずタイピング技能は必須です。一般ユーザは、この技能が一つの関門です。タイピングは両手を使いますので、指がキーボードのホームポジションから外れる使い方を嫌います。マウスを使うときは手が離れますので、マウスを使わない方法としてショートカットキーを愛用します。テキストエディタやワープロには編集(EDIT)のメニューがありますが、「カット・コピー・貼付け」は Ctrl キーを押しながら X,C,V キーを打つことで代行できます。この指使いは、英大文字を打つときに Shift キーを押しながら文字キーを打つ操作の拡張です。したがって、Ctrl キーは、キーボードの左右にあるのがプロ用の仕様です。また、マウスに代えて、タッチパネルを装備するノートパソコンが増えてきましたが、手がキーボードから離れない使い方ができます。このようなことを考えて、"Prototype"には標準的な「カット・コピー・貼付け」のメニュー項目は意図的に省いてあります。

Windows は基本的にキーボードからの制御をサポートしている

パソコンの OS である Windows は、マウス無しでも総てキーボードで操作できるように設計されています。その使い方は Windows のヘルプでショートカットキーを引けば調べることができます。Microsoft 社が関係しているソフトウエアはかなり徹底していますが、一般的なアプリケーションではマウスだけしか使えない制御方法も少なくありません。"Prototype"でもキー操作を主体とすることを意識しましたが、パネルのクリックなどはマウス操作だけです。標準的に使うショートカットキーは、念の為、下に示します。

ショートカットキー	英語の処理名	処理の内容
Ctrl+A	select all	全部を選択範囲とする
Ctrl+Z	undo	直前の編集処理の取り消し
Ctrl+X	cut	切取りまたは削除
Ctrl+C	copy	選択範囲をコピー
Ctrl+V	paste	指定した場所に貼付け

クリップボードを効率的に利用すること

編集機能の「カット・コピー・貼付け」は、Windows がサポートしている**クリップボード**を介して行いますので、並列して幾つかのアプリケーションとの間で、文字とグラフィックスとデータの受け渡しができます。Windows の環境では、複数のアプリケーションを擬似的に並列に実行できます。文字データは子ウインドウのテキスト出力画面を文字データのバッファーに使い、必要であれば WORD を開いてデータの移し替えができます。グラフィックスの場合に、グラフィックスソフトとの間でデータの移し変えができます。グラフィックス関係のソフトウエアは種類が多く、またデータのメモリを多く使いますので、作業用にデータをファイルに書き出しておいて、後で図の編集をするのが勝ります。Windows には Microsoft のユーティリティとしてグラフィックスソフトがサービスされていますので、それを使うとよいでしょう。ただし、Windows のバージョンによってソフトウエア名に違いがありますので、使い方を確かめておきます。Windows2000 では、Microsoft Photo Editor、WindowsXP では Microsft Office Picture Manager です。これらを利用する主な目的は、メモリを大量消費するビットマップイメージを gif 形式に変換することに使います。ウインドウズのデスクトップ全体を画像としてクリップボードにコピーする方法は、PrtScr キーを使うことができます。このキーの使い方は、パソコンのキーボード配列によって、違いがあります。クリップボードの中身を生で見たいときのユーティリティは、クリップボードビューアがあります。clpbrd.exe の名前です。

1.9 デバイスドライバの概念

OS はプログラムと装置とを繋ぐ

オペレーティングシステム(OS)は、パソコンに常駐している一つのソフトウエアです。この機能は、ディスクを始め、種々の装置（デバイス）とのインタフェースを調整することが主な目的です。コンピュータはデータを読み書きする外部装置を必要とします。遥か昔は、コンソールタイプライタが標準の入出力装置でしたが、カード・紙テープ・磁気テープ・ディスク・プリンタ・モニタ・プロッタなどの多種類の入出力装置が開発されてきました。プログラミングでは、装置が変るごとに、その装置に合わせて入出力文を書き変えると手間が大変ですので、プログラミング上では、装置の違いを番号などで指定すれば済むような工夫が考えられました。DOS は、Disk Operating System の頭字語です。入出力装置すべてを、擬似的にディスクと見なし、装置の違いは番号や記号で表すようにします。文字データの読み書きのプログラム文は単純になり、例えば、Basic 言語では、基本的なキーワードの INPUT と PRINT を使う文で済ますことができます。OS は、装置側で読み書きに使うプログラム文との間にソフトウエアのデバイスドライバを介在させるようにします。そうすると、装置を取り替えるときは、デバイスドライバを入れ替えます。このソフトウエアは、装置のメーカー側が提供しますので、新しい装置を繋ぐときには、その装置用のデバイスドライバを繋ぐ予備作業が必要です。

グラフィックスは標準化が難しいこと

コンピュータで図形を扱う処理は、一言でコンピュータグラフィックスと括りますが、大きく分けて、線図と濃淡図とで別扱いです。線図の方は、プロッタを利用した自動製図で使う技法と関係しました。線の太さや色を変えるのは、装置側でペンを取り替える方法で対応させますので、多様な色遣いを工夫した作図には向きません。線図は、或る範囲を塗り潰す濃淡図に代えて、ハッチング(hatching)技法を使います。濃淡図は新聞写真を拡大して見れば分かるように、細かな点（ドット）の疎密で表現します。線もドットの並びで表しますので、ドットの密度が粗いと、細くて滑らかな線を表すことができません。グラフィックスモニタはドットの集合体で図形を表しますので、眼を近づけて見るような細密画には向きません。Windows の環境では、濃淡図の方のグラフィックス機能に重点がありますので、線図をサポートする機能が充分ではありません。モニタは、最初からパソコンと接続して使うグラフィックス装置です。プリンタやファックスはメーカー別にデバイスドライバを組み込むようになっています。プログラミング上は作図の命令文が必要ですが、INPUT 文、PRINT 文なみの標準化は進んでいません。したがって、この部分に関して"Prototype"にグラフィックス用のデバイスドライバの考えを使った関数をプログラミングして、Windows 側のグラフィックス機能を間接的に利用するようにしました。

内部メモリを効率的に使う方法

メモリはコンピュータのプロセッサ(CPU)から見るとき、内部メモリと外部メモリに分けます。外部メモリはディスク装置などですので、そのデータを一旦内部メモリに移して処理します。ディスクとのデータの入出力は、CPU の処理速度に較べて遅いのと、その分だけ内部メモリの領域を圧迫します。ここでも、ディスク利用のテクニックが応用されています。OS はコンピュータに常駐しますので、便利さを追求してこの寸法を大きくするとユーザのプログラム領域を圧迫します。これを抑えるためと、作業時のメモリ領域を能率的に使う方法が二つあります。一つは、ユーティリティプログラムとして処理を独立させる方法です。Microsoft 社が提供するユーティリティは、おまけとして最初から付いているものと、別途購入しなければならないものがあります。WORD や EXCEL は後者のソフトウエアですが、パソコンを購入するとサービスとして組み込み済みが多くなりました。もう一つは DLL (Dynamic Link Library)のように、必要になったときに内部メモリに読み込んで実行させる方式です。DLL は、Windows 側で準備したもの以外に、私的にも作成されますので、*.DLL でファイル検索をすると数千個も見つかります。同じ名前も多数見つかりますが、バージョンが違う場合もありますし、偶然同じ名前であったりします。この中身がブラックボックスですので、Windows の理解を難しくしている原因となっています。メモリ領域が小さな時代のコンピュータでは、overlay と言う技法を使いました。マイコン用の Basic は CHAIN 文を使いました。この方式の延長が DLL であると考えるとよいでしょう。商品として作成するソフトウエアは、あれもこれもと欲張って多機能化する傾向があります。科学技術計算では、なるべく変数領域を大きく取れるようにし、できればプログラム単位を小さくして、見通しを良くし、バージョン改訂が大きな負担にならないようにします。

1.10　CUI の操作方法の利点

リダイレクションの機能が欲しいこと

DOS の環境にあるパソコンで便利であった機能の一つがリダイレクション(redirection)です。DOS は、総ての入出力デバイスを抽象化してディスクとして扱います。キーボードもプリンタも、拡張した概念でディスクの扱いでできます。そうすると、装置（デバイス）名または装置の番号を変えるだけで、キーボードからの文字入力をディスクからの入力に切り換えることができました。例えば、対話形式で実行するプログラムは、ユーザがキーボードからデータを入力して処理を進めます。そこで、この入力データをテキストファイルに作成しておいて、それを読み込ませることに代えることができます。例えば、下のように書くことができました。

TYPE A:DATA.TXT >*PRN*	(1)
COPY CON, A:TEST.TXT	(2)
someprogram < A:DATA.TXT	(3)

書式(1)で、*TYPE* は MS-DOS のコマンドですが、引き数を持つプログラムと考えることができます。A:DATA.TXT は、ディスク A に在るテキストファイル名です。ここまで書くと、ファイルのリストがモニタ画面にリストされます。>*PRN*がリダイレクションの指定であって、リストをモニタではなく、プリンタに書き出します。*PRN*は、プリンタを表す定義された装置名です。書式(2)の *COPY* も MS-DOS のコマンドです。引き数を二つ *CON* と A:TEST.TXT を使いますが、意味はファイル *CON* をコピーして、フアイル A:TEST.TXT を作ります。この *CON*はコンソール装置の定義です。これを実行すると、ユーザのキー入力待ちに入り、入力されたデータがテキストファイルとして書き込まれて行きます。つまり、*CON* はディスクファイル扱いになります。ファイルの終りは、Ctrl+Z を書き込みます。これを入力するとコピー作業が完了します。書式(3)は、キーボードからデータ入力を受けつけて動作する対話型のプログラム実行を、キーボード入力に代えてファイルからの入力に切り換えます。バッチファイルの中にこの形式を含ませると、一種の自動実行ができますので、ユーザはコンピュータのキーボードの操作から解放されます。プログラムの切りの良いところで、適当な待ち時間を設けると、デモンストレーションができます。Windows の環境で、マウスを使って制御するソフトウエアは、このリダイレクションができませんので、もしデモンストレーションをさせたいとなると、別にビデオを作成するか、マイクロソフトのパワーポイントを使うか、などの別作業が必要です。

マニュアルの編集方法も変ること

リダイレクションの機能が使えると、入力に使うテキストファイルそのままを例題として編集することができます。例題用にテキストデータを準備して実行させますと、どのように処理が進むかが分かります。また、例題のデータを書き変えれば、別の条件の計算がすぐにできます。プログラムの説明は、操作方法ではなく、データの準備方法と結果の見方に焦点を置くことができます。Windows の環境で機能するプログラムの場合には、作業画面全体の図形を表示して、マウスでクリックする箇所が画面のどこにあり、どれを選択するかなどの操作方法を説明しなければなりませんので、多くの図版を使わなければなりません。ページ数が多い割には中身の文章説明は多くありません。

ログファイルの作成と利用ができること

設計計算に使うプログラムを DOS の環境で対話型の実行をするときは、計算の途中経過を見ながら作業を進め、もし計算結果が条件に合わなければ、前に戻ってデータを変えて試行します。このような方法が試行錯誤(trial and error)です。このとき、入力作業のテキストデータを総てファイルに書き出すと、どのように作業が進められたかを確認することができます。このような記録に使ったファイルをログファイル(log file)と言います。ログファイルを編集して、手戻りの箇所を削除すると、最初から無駄のない処理の進め方の記録が残ります。この編集されたログファイルを、バッチ処理用の入力ファイルにできれば、無駄を省いた実行を再現することができます。似たような実行形態は、碁や将棋のゲームプログラムで見ることができます。手順を記録する機能がログファイルです。ゲームの進行では、「待った」を掛けて処理を戻すことができます。設計計算の場合も、形は違いますが、処理を前に戻すことはごく普通に行われますので、「待った」と同じ考え方をしています。CUI の環境では、記録と再現とにテキストデータの利用ができる便利さがあります。

2. ファイルの構成

2.1 プログラミングの作業全体

統合開発環境を構成すること

一つの実行形式のプログラム（ここでは**アプリケーション**と言いかえます）を作成するとき、幾つかの作業段階を経由し、その段階ごとに固有のプログラムを使います。具体的に説明すると、最初に、採用するプログラミング言語を決めます。それに合わせて、**テキストエディタ**で**ソースコード**を編集します。このファイルを読んで、**オブジェクトコード**に翻訳します。これに使うプログラムが**コンパイラ**であって、プログラミング言語に固有です。ここでもオブジェクトの用語が出てきますが、英文法で言う目的語と同じ意義で使っていて、**対象とする CPU が理解する機械語のコード**と言う意味です。オブジェクトコードは、どのコンパイラ言語で作成したかは関係が無くなっています。幾つかのオブジェクトコードを結合させて(link:リンクと言います)、実行形式のアプリケーションを完成させます。そのときのプログラムが**リンカー**です。この他に、作成段階のアプリケーションを検査する**デバッガ**なども使います。この作業は入り組んでいますので、プログラミング全体の作業環境を整えるアイディアが生まれました。これが統合開発環境 IDE (Integrated Development Environment)です。プログラミング言語のベンダー（販売会社）は、一つの実行形式のプログラムで提供しています。Visual Basic 6.0 の場合には VB6.exe がそうです。この中から、プログラミング作業に必要な種々のプログラム（テキストエディタ、コンパイラ、リンカーなど）を間接的に利用します。

プロジェクトと言う概念を使うこと

アプリケーションを作成する作業全体を、総括的に**プロジェクト**(project)と言うようになりました。少し硬い日本語は事業計画と訳します。開発したい**プログラム名**と関連をつけて、**プロジェクト名**を付けたフォルダ（**プロジェクトフォルダ**）を準備することから始めます。このフォルダの中に、作業に関係するファイルを納めます。これらのファイルの目録に相当する特別なテキストファイルがあって、**プロジェクトファイル**と言い、VB6 では拡張子(.vbp)が付きます。このファイルは、マイクロソフト社の提供する VB6 の開発ツール IDE が作成し、管理します。したがって、或る作成済みのプロジェクトファイル名をクリックすると VB6 の IDE が立ち上がります。プロジェクトファイルは、テキストファイルであって、プログラミングに必要な関連ファイル名などが書き込まれています。したがって、この中身はテキストエディタで開いて見ることができます。プロジェクトファイルの名前は、作成するアプリケーションのプログラム名(.exe)に転用することを考えておきます。**フォルダ名**も、プロジェクト名をそのまま使うのが分かり易いでしょう。プログラミング作業を始める前に新しいフォルダを作成しておきます。プログラミング作業の早い段階で、プロジェクト名を決定し、プロジェクトファイルを作成します。これは、IDE のファイルメニューで「名前を付けてプロジェクトの保存」を実行すると作成できます。ソースコードレベルで VB6 のプログラムを配布するときは、プロジェクトフォルダ単位で行います。別のフォルダに在るファイルを参照していると、そのプロジェクトに必要なファイルが含まれませんので注意が必要です。

モジュール名とファイル名の区別が必要

プログラマがする主要な作業は、テキスト形式のソースコードを準備することです。これは、管理を便利にするため、幾つかのファイル（**モジュールファイル**）単位に分けて作成します。VB6 では、その中身よって二種類のファイルがあります。拡張子(.bas)の付く**標準モジュール**ファイルと、拡張子(.frm)の付く**フォームモジュール**ファイルの二種類です。前者が、普通に言う Basic 文で書くプログラム文です。後者は、Windows の環境で動作するアプリケーションを作成するようになって必要になったファイルであって、この作成作業が、オブジェクト指向プログラミングの主題となるものです。これら二種類のテキストファイルは、VB6 の IDE の管理下で使いますので、中身が見えない部分があります。この部分は、Windows の機能を使う固有のソースコードであって、ユーザが勝手に書き変えると正しく機能しなくなる危険があるからです。モジュールファイルの中身を表すキーワードを**モジュール名**と言い、プログラミングコードの中では重要な名前です。**ファイル名**は、ファイル管理に使う名前ですので、モジュール名と同じにする必要はありません。違うファイル名であっても、モジュール名が同じであるとエラーが起きます。特にフォームモジュールのモジュール名は、フォーム(Form)そのものの宣言名ですので、その命名には注意が必要です。

2.2　ソースコードの編集作業

ファイルの参照は注意が必要であること

何かのアプリケーションの開発をするとき、最初の作業は、専用のフォルダを新しく作成し、新しいプロジェクト名を決めます。IDE のファイルメニューから、名前を付けてプロジェクトの保存を指定して、最初の作業環境を作成します。今、別のプロジェクトが既にあって、その改訂版や姉妹編のアプリケーションを作成したいとします。そのプロジェクトの IDE 作業環境から、新しいフォルダに「名前を付けて保存」処理をすると、そこに新しいプロジェクトファイル(.vbp)が作成されます。そのとき、プロジェクトを構成するファイルは、元のフォルダ内にあって、そちらを間接的に参照しています。プロジェクトファイルは、利用するモジュールファイルがどこにあるかのディレクトリ情報が書いてあります。つまり、別の場所にあるモジュールファイルを参照して使う方法があって、その宣言は IDE からモジュールの追加メニューで行うことができます。しかし、当該のプロジェクトでモジュールの中身を書き変えると、元のモジュールファイルを書き変えることになりますので、元のプロジェクトを壊してしまいます。この危険を避けるため、元のファイルの**プロパティ**（属性を表す定数などのこと）を読取り専用にしておきます。しかし、最も安全な方法は、フォルダごとコピーして新しい作業用のフォルダを作ることです。部分的に外部のモジュールファイルを使いたいときは、そのファイルを作業用フォルダ内にコピーして参照するのが安全です。そうすると、同名のファイルができますので、その区別が付くように、名前の後に 1,2…などを付ける、などの注意が必要です。

標準モジュールのコーディング

VB6 の二種類のソースコードのうち、標準モジュール(.bas)のソースコードの方は、以前の DOS 時代の Quick Basic, N88Basic などと基本的に同じ構文です。したがって、コーディングの作成技法は、以前のソースコードの場合と殆ど同じですし、また以前のソースコードを多少手直しすればそのまま利用できます。この部分のプログラミング技法は、開発したい専門分野（例えば筆者では橋梁工学）に固有する習慣がありますので、以前の BASIC プログラミングの知識を勉強しておく必要があります。モジュールの中身はテキストファイルですので、別のテキストエディタで作成したファイルを拡張子*.bas に付けかえると、そのままプロジェクトの標準モジュールに追加することもできます。例えば、Fortran のソースコードを使いたいときがそうです。N88Basic 用のソースコードは、同じ拡張子*.bas の標準モジュールとして使うことができます。しかし、IDE が内部的に利用しているテキストエディタは特殊であって、そのままで文法チェックなどが行われます。Fortran もしくは N88Basic の文法は VB6 と違いますので、エラー表示が出ます。この修正のヒントは、後の５章と６章で解説します。

フォームモジュールのコーディング

フォームモジュールは、パソコンのオペレーティングシステムが Windows**に変更されたことで必要になった新しい種類のソースコードです。DOS 時代のソースコードを Windows 版の VB6 に書き換えるときには、フォームモジュールを新しく追加します。上で説明した標準モジュールを別に作成しなくても、そのソースコードをフォームモジュール内に組み込んで済ますこともできます。フォームモジュールの場合には、最小プログラム単位である**プロシージャ**（副プログラム、サブプログラム、ファンクションなどとも言います）には、オブジェクトと関係して IDE が発行する名前がありますので、勝手に追加や変更ができない名前があります。したがって、モジュールの追加などは、プロジェクト管理下のモジュールの骨格を作成表示しておいて、クリップボード経由でテキストの貼付けをするのが安全です。フォームモジュールのプログラミングが、いわゆるオブジェクト指向プログラミングです。概念的に言えば、コンピュータの中に擬似的なハードウェア部品（オブジェクト）を作成し、それを制御するプログラムを書くことです。最も基本的なオブジェクトは、DOS で利用していたモニタ画面をモデル化したウインドウ図形です。これを**フォーム**(form)と呼ぶことから、フォームモジュールの名称が使われます。フォームモジュールのソースコードの大部分は、種々のオブジェクトの設計データ（プロパティと総称します）を記述したテキストです。しかし、IDE の作業画面では、プログラマの作業性を良くするため、この設計データの記述部分を表示しません。それに代わる方法として、IDE は、プログラマがフォームの幾何学的形状設計を GUI の環境で構成できるようにしてあります。オブジェクトの詳細は、IDE のプロパティウインドウの表項目で定義します。プログラマは、フォームモジュールのテキスト全体がどのように構成されているかの知識がある方が望ましいので、フォームモジュールを別のテキストエディタで開いて覗くとよいでしょう。

2.3 イベントプロシージャ

システム側が自動発行する

プログラマが作成するソースコードは、標準モジュール、フォームモジュール共に、プロシージャの集合です。しかし、フォームモジュールの方は、あらかじめ VB6 の方で決めた名前で自動発行される**イベントプロシージャ**を扱うことが大きな特徴です。言い方として、イベントハンドラー、**コールバック**関数(Callback)などがあります。プログラムのユーザが、マウスを使ってクリックするなど、オブジェクトを操作したとき、システムがこれをイベント（事件が起きた！）と認識し、「何のご用でしょうか」と問い合わせをしてくる窓口がイベントプロシージャです。そのときに何をさせるかのソースコードをプログラマがイベントプロシージャの中に記述します。イベントプロシージャはサブルーチンの形をしていて、これを呼ぶのはシステム側ですし、処理が済んで復帰する先もシステム側です。この遣り取りは、電話で相手を確認する方法と似ていますので、通称で **Callback** と言います。この部分は、システムが呼び、帰りもシステムですので、ブラックボックスです。Callback も、声は聞こえても相手を眼で確認する手立てがないのと似ています。これが Windows 関連のプログラミングを面倒にしています。イベントプロシージャの中で使う定数などが**プロパティ**であり、ステートメントに相当する命令を**メソッド**と言い換えます。イベントプロシージャの中に何もコードを書かなければ、イベントそのものがシステムで無視され、また、プロジェクトを保存するとき、そのイベントプロシージャ自体も削除されます。イベントプロシージャの中に、プログラマがコードを書き込むことがプログラミングです。この他に、プログラマが Basic 流のプロシージャを別に作成し、そこに処理を回すようにもできます。同じフォームモジュールの中にプログラマが独立にプロシージャを作成することもできますが、別に作成する標準モジュールの方にプロシージャを作成しておいて、それを呼ぶようにすると弾力的なプログラミングができます。また、モジュール間を横断的に利用する共通変数は、どこかの標準モジュールで宣言しておきます。

フォームまたはオブジェクト名で修飾して使う

フォームモジュールの中のプロシージャで書くプログラム文は、標準的な Basic 流のステートメントです。入出力に関係するプログラム文を書くとき、例えば Print 文では、書き出す対象のオブジェクト名を明示的に示す必要があります。例えば、「FormConsole.CMonitor.Print …」のようです。修飾の方法は、VB6 では点(.)を使いますが、プログラミング言語によって違いがあり、C++では -› と書きます。日本語で理解するときは、「何とかの何とかの何とか…」と読み換えるとよいでしょう。オブジェクトの寸法は、Width, Height のような標準的なキーワードを使います。これは VB6 側で定義した変数名であって、総称が**プロパティ**です。日本語を当てるなら属性です。変数と同じように型があり、代入や読み出しに使います。これも、対象とするオブジェクト名の修飾が必要です。Print 文は、VB6 では Print メソッドと呼び換えます。対象としたオブジェクトに何かの処理をさせるプログラム文単位は、サブルーチンやファンクションの性格がありますが、これも**メソッド**と呼びます。他の代表的なメソッドは、グラフィックスで線を描くときの Line メソッドです。Quick Basic などでは Line 文と言いました。

文字入力とマウス処理を認識するイベントプロシージャ

DOS の環境で動作するパソコンの初期画面では、ユーザのキーボードからの入力文字は、すぐにモニタに表示され、同時にシステム側でその文字列の意味を解読して、それなりの処理が行われます。Windows の環境では、キーボードからの文字入力に対して何の反応も示しません。マウスで何かのアイコンを選択してクリックするか、Ctrl キーなどの特殊キーを使った特別なキー操作に対して反応します。VB6 で新しいプログラミングをしたいとして、IDE の初期画面を開くと、標準で一つのフォームを準備してくれます。何も手を加えないで、そのまま実行させると、タイトルバー付きの一つのフォームが表示されますが、マウスにもキー入力にも全く反応しません。マウスで制御するタイトルバーの寸法変更ボタンとプログラム終了の閉じるボタンはデフォルトで生きています。タイトルバーの左にあるシステムアイコンも生きていて、同じ処理ができます。プログラミングの最初は、とにかく、キーボードの入力とマウスのクリックに対して反応するイベント処理を追加しなければなりません。VB6 の入門書は、あらゆる可能なイベント処理が細かく解説してありますが、そもそも、何を目的としてプログラミングをするか、の焦点がはっきりしない説明です。数値計算のように明確な目標があるプログラミングでは、イベント処理の部分に必要最小限の骨格を作成しておくことにします。これがプロトタイプです。最小限と言っても、かなりの分量のコーディングを準備しなければなりません。

2.4 プロトタイプ用フォームモジュール

フォームの図形情報はすべて文書で記述される

デスクトップに表示されるウインドウ図形のデータは、総て文書の形でフォームモジュールに記述します。グラフィック化された図形データではありません。その文書を OS である Windows が解読して図形に描き、また種々の制御の手助けをします。図形データを生で使うのは、アイコンなどの図柄です。これらをリソースデータと言います。文字のフォントなどもリソースデータです。標準的なリソースデータは Windows が持っていますので、なるべくそれらを利用するようにすると、ソースコードを軽くすることができます。

プロトタイプのモジュール

VB6 によるアプリケーションの開発は IDE の管理下で行います。新しくアプリケーションを作成する新規プロジェクトは、デフォルトで一つの空白のフォームモジュールを準備してくれます。新たに複数のモジュールファイルを準備するときは、IDE のプロジェクトメニューから、新しい**モジュールの追加**をします。これらのモジュールに最初から個別のソースコードを書く煩わしさを避ける方法として、見本になるモジュールを他のプロジェクトファイルからコピーして手直しするのが実践的です。この方法は、プロジェクトメニューの**ファイルの追加**で行います。このとき、比較的応用の利く雛型（プロトタイプ）のモジュールを準備しておこう、というのが**プロトタイプのモジュール**を提案する意義です。ここで言うプロトタイプは、ソースコードレベルでの一種のライブラリであって、実行時（ランタイム）に参照される DLL、もしくはコンパイル時にマシン語レベルで結合するコードではありません。プロトタイプのモジュールはファイル単位でコピーして追加しますが、中身を部分的に書き換えることがありますので、できるだけ元のファイル名を避けます。同種のファイル名は、例えば、接尾文字として 1、2...などを付けて区別するようにします。

モジュール名とファイル名

フォームモジュールは、いわゆるウインドウ単位で作成します。モジュール名はプログラム文の中で共通名扱いです。ファイル名の方はソースコードの管理に使います。全体は管理用に固有の名前を付けたフォルダに納めますが、フォルダ違いでも同名のファイルを使うことがあります。全く同じファイルであるのが望ましいのですが、姉妹関係で少し中身が違うことがありますので、ファイル名の末尾に 1,2...のような添え字を付けて区別するなどの注意が必要です。"Prototype"では、下のように、モジュール名とファイル名は 5 種類考えます。

表 2.1. フォームモジュールプロトタイプ：モジュール名とファイル名

番号	モジュール名	ファイル名	用　　途
f1	MDIFormConsole	[ConsoleMDI1.frm]	親ウインドウ
f2	FormConsole	[Console1.frm]	コンソール画面表示用子ウインドウ
f3	FormLprint	[Clprint1.frm]	プリンタバッファ用子ウインドウ
f4	FormGraphic	[Cgraphic1.frm]	グラフィックス表示用子ウインドウ
f5	FormAboutBox	[CaboutBox1.frm]	バージョン表示用ウインドウ

フォームモジュール名は共通名であること

フォームモジュール名は、フォーム、つまりウインドウに付ける固有の名前です。別のモジュールから、そのフォームのオブジェクト・メソッド・プロパティを参照するときの修飾名になります。フォームおよびそこで使う種々のオブジェクトのプロパティは、フォームモジュールの中にテキストとして記述されます。その書き方は、VB6 の IDE が自動作成し、プログラマが直接に作文しないようになっていて、IDE のテキストエディタでは見えないように隠しています。プロパティのデータは、IDE の画面でプロパティウインドウを表示しておいて設定しますが、そのデータは内部でフォームモジュールのファイルに書き込まれます。したがって、データを変更するとファイルが部分的に書き変えられます。見えているコードが同じであっても、隠されている部分のコードが違うことがあります。このこともあって、同じファイルを別のプロジェクトで相互に共用することを避けます。同じファイルを別のプロジェクトでも使う場合には、書換えの危険を避けるため、ファイルの属性を読取り専用にしておきます。

2.5　プロトタイプ用標準モジュール

標準モジュール

フォームモジュールでは、フォーム（ウインドウ単位）のユーザインタフェースの窓口（Callback）を記述しますので、込み入った処理は、そこから標準モジュールのプロシージャに制御を渡すように計画します。したがって、標準モジュールの幾つかは、フォームモジュールと組にして利用します。この場合、モジュール名・ファイル名・プロシージャ名に、あらかじめ名前の付け方を決めておくことにしました。標準モジュール名は、プログラム内部で使うことがありません。ファイル名の方は、プロジェクトごとに違いますので、表 2.2 に示すように添え字を付けて区別できるようにしておきます。フォームモジュールから参照するプロシージャには定型的な名前が幾つかあります。例えば、プログラム開始のプロシージャ名は **Main** を使います。ただし、この中身はプロジェクトごとに固有です。このプロシージャを含ませる標準モジュール名は MainProgram とします。このファイル名（.bas の付くファイル）はプロジェクトごとに固有のファイル名です。標準モジュールはプロシージャ単位を集めたものですので、制御の入口は必ずしもモジュールの先頭ではありません。プロシージャ名については、後の章で個別に解説します。

表 2.2　標準モジュールプロトタイプ：モジュール名とファイル名

番号	モジュール名	ファイル名	用　　途
b1	N88Sub	N88Sub1.bas	NEC-PC98 の画面エミュレータ
b2	VBGsub	VBGsub1.bas	標準グラフィックスサブルーチン
b3	ConsoleIO	ConsoleIO1.bas	コンソール入出力用基本サブルーチン
b4	CommonMenu	CommonMenu1.bas	共通に利用するメニュー処理の集合
u1	MainProgram	*****.bas	Public sub Main()を含ませる

3. フォームモジュールの解説

3.1 MDIFormConsole

プログラムの開始と終了

何かのプログラムを実行させる最初を、俗に**立ち上げ**(boot, startup)と言います。Windows の環境で動作するプログラムは、立ち上げると少なくとも一つのフォームを表示し、ユーザとパソコンシステムとのインタフェースが図られます。プログラムの終了は、DOS の環境では、終了方法もプログラミングする必要がありました。常識的な方法は、キーボードから何かのコマンド (例えば QUIT)を入力します。ショートカットで、Ctrl+Q と打つ方法もよく使います。ただし、そのように機能するプログラミングが必要です。Windows の環境のプログラムは、標準としてフォーム右上の閉じるボックスをクリックします。親フォームの左上のシステムアイコンをクリックして表示されるプルダウンメニューからも終了が選べます。FILE メニューに終了サブメニューを組み込むようにプログラミングすることもあります。幾つもの方法がありますが、必ずしも終了しないことがありますので、終了方法を確かめておく必要があります。プログラムの強制終了は、Ctrl+Alt+Del キーで行います。余談ですが、コンピュータが**ハングアップ**したとき、キーにもマウスにも反応しなくなることがありますので、電源コードを抜いて、強制的に電源を切ります。ノートパソコンでは電池でバックアップしていますので、電源コードを抜いても電源が落ちません。電源スイッチのキーを押しっぱなしにすると切れます。

エクスプローラ形式の画面デザインは使わない

Windows のアプリケーションは、実行時画面に定型的なスタイルの提案があって、MDI・SDI・エクスプローラなどがあります。この細部デザインにはプログラマの好みが反映します。インターネットが普及していますので、多くのユーザは、エクスプローラのスタイルに馴染んでいます。しかし、このスタイルは採りません。このスタイルは、多様な選択肢を限られた狭いモニタ画面に、できるだけ多くの項目を表示してランダムに選ばせる方式です。多くの項目は、表形式で並べてあるのですが、表の枠は見えないようにしています。項目を目立たせるために、カラーやグラフィックを多用して派手なゲーム画面にすることが多いようです。科学技術計算のように、筋書きのある作業を支援する目的の画面には向きません。筋書きのある作業を補助する概念が**ウイザード**です。

プログラムの入口を設計する

プログラム文を作成するとき、システムからの最初の入口がどこにあって、どのようにプログラム文を辿るように処理が進むかを意識しなければなりません。一般的には、標準モジュールの中で main をキーワードに持つプロシージャ、Public Sub Main()に最初の制御が入ります。小さな VB6 のアプリケーションでは、標準モジュール無しで、フォームモジュールのイベントプロシージャにコードを書くだけで済ますことがあります。このとき、システムの制御はデフォルトでフォームモジュールに入ります。その入口は、隠されて見えませんが、内部的には WinMain()のプロシージャを呼んでいます。そこでの必要な処理が済むと、制御がイベント待ちに入ります。"Prototype"では、この最初の部分を明示的にプログラミングします。VB6 の IDE の作業画面で、プロジェクトメニューからプロパティを選びます。そこの全般にあるスタートアップの設定を MDIFormConsole に指定します。そうしておくと、プログラムが立ちあがると、システムからの制御は MDIFormConsole のイベントプロシージャ Form Load に入ります。そこから Sub Main に制御を渡すようにコードを書きます。Main から帰ると、制御がシステムに戻り、イベント待ちに入ります。なお、下のリストでは、StatusBar のイベントプロシージャも示してあります。これも制御を標準モジュールに作成した PanelClick で細かな処理を行わせます。

```
Private Sub MDIForm_Load()
    Call Main
End Sub
'--------------------------------------------------------------------
Private Sub StatusBar1_PanelClick(ByVal Panel As MSComctlLib.Panel)
    Dim Vpanel As Variant
    Vpanel = Panel
    Call PanelClick(Vpanel)
End Sub
```

フォームの幾何学的なデザインと表示

MDIFormConsole は、親ウインドウの機能で設計します。VB6 の IDE の作業画面では、最初に一つの Form を準備してくれますが、これは、子ウインドウの仕様に変更することができます。親ウインドウは、プロジェクトメニューを開いて、MDI フォームモジュールの追加で新規に作成することになっています。この親ウインドウは一つしか定義できません。この外形は、第１章の図 1 を見て下さい。この親ウインドウの寸法などは、設計時にプロパティウインドウで設定します。モジュール名は MDIFormConsole、ファイル名は MDIFormConsole?.frm としました。この親フォームに載せるオブジェクトは、ステータスバーと、ファイルやフォントなどの設定に利用するコモンダイアログボックスの二つです。プログラムを実行すると、システムは親ウインドウの設計時情報を元にしてデスクトップにフォームの図形を表示して制御が Form_Load に入ります。そこから Sub Main に制御が渡ります。このプロシージャは標準モジュールの MainProgram の中にコーディングしてありますが、説明のために下にリストの一例を示します。

```
  Public Sub Main()
      InitWindow
      Load FormGraphic
      Load FormLprint
      Load FormConsole
      MDIFormConsole.MDIDialog.InitDir = CurDir
      MDIFormConsole.FileSaveDialog.InitDir      =
CurDir
      InitializeData
      HtmlHelpFilterName$ = "DataIntegral.chm"
      CRT01
  End Sub
```

子ウインドウは幾何学的デザインを設定してから表示する

ウインドウの位置と寸法は、ウインドウを表示した(Load)後でも変更することができますが、Load の前に寸法を決定しておきます。InitWindow は、図１のように子ウインドウを並べる位置と寸法を指定し、それを元に、三つの子ウインドウ(FormGraphic, FormLprint,FormConsole)を Load します。この他に、幾つかの初期設定を済ませてから、CRT01()を呼びます。これはコンソール画面に最初の案内表示をさせるプロシージャであって、その表示例が図１の左上です。CRT01 から帰ると、制御がシステムに帰り、プログラムはイベント待ちに入ります。つまり、ユーザがパネルをクリックするか、メニューの選択を待ちます。"Prototype"のステータスバーのパネル割りは、左から【戻る】【1】【2】【3】【4】【OK】、としました。メニューの機能は、個別に子フォームの解説の箇所で説明します。

オブジェクトの宣言文は隠されて見えない

ステータスバーは、これを幾つかのパネルに分け、キーボードのファンクションキーと似た使い方をプログラミングします。ステータスバーの基本的なプロパティは、IDE のプロパティウインドウで設定できますが、パネル割りの設計は別のプロパティページを表示して作業します。この設定値は、裏でフォームモジュールのファイルに書き込まれます。見えない部分のリストを確認したいときは、テキストエディタでフォームモジュールを開きます。ウインドウの画面では、メニューと共に、このパネルをマウスで選択してクリックすることで制御をシステムに伝えます。MDIFormConsole のソースコードは、コールバックのプロシージャを二つ使います。一つはプログラム全体の入口、もう一つはコンソール画面の切り替えに利用するステータスバーの、パネルクリックに応答する処理用です。パネルをクリックしたときに処理を引き継ぐ PanelClick プロシージャは、Main と共に MainProgram モジュールに含ませます。プログラミング全体の骨格は、殆ど PanelClick プロシージャにあります。もしプログラム全体像をフローチャートに書くならば、このプロシージャを対象とします。メニューの選択による処理は、多くの場合、ランダムに選択できる性格を持つ処理に当てます。

3.2 FormConsole

一行分の文字入力枠を常備する

オペレーティングシステム(OS)が DOS から Windows に変化したことで最も影響を受けたことは、キーボードから文字を入力してシステムに何かの指示を与える方法(CUI)が面倒になったことです。FormConsole 子ウインドウは、古典的な DOS のコンソールの機能を持つように設計しました。これは、キーボードの入力を受け付ける一行幅の**テキスト入力枠**と、プログラム側からのメッセージを表示する**コンソール画面**とで構成します。テキスト入力枠のモデルは、EXCEL の数式入力バーと同じアイディアです。VB6 では親ウインドウのオブジェクトとして扱うことができませんので、この子ウインドウの上部に載せました。メニューバーは、実行時には親ウインドウに所属するように表示されますが、メニュー項目の定義とイベント処理は、所属する子フォームモジュールのコード部分で定義します。

DOS の画面を再現するデザインであること

子ウインドウの画面（フォーム）は、親ウインドウの作業域（クライアント領域）の中に入り、また子ウインドウにもタイトルバーなどが付きますので、子ウインドウの実効作業域は相対的に高さが低くなります。子ウインドウのクライアント領域に Print 文で文字を書き出すこともできますが、文字入力を受け付ける**テキスト入力枠**がクライアント領域を部分的に隠しますので、独立した**ピクチャーボックス**を一つ用意し、これを定型的な用紙のように考えて、ここに文字や図の配置をデザインし、フォームに左上詰で最大に接するように載せます。この用紙領域は、DOS のコンソール画面をエミュレートして（模して)、640×400 の画面寸法比と相似な画面を設計します。オブジェクト名は、**CMonitor** と命名しました。ここに、半角で 80 文字×20 行の表示を考えます。この寸法系列は、N88Basic の画面をエミュレートしますが、DOS の標準画面比 640×480 より高さを低く設定することになります。このピクチャーボックスの割り付けは、標準モジュール N88Sub1 にまとめました。

案内表示はプログラムステータスで切り替えて表示する

CMonitor に表示する案内表示は、定型的な用紙寸法に書く書式で準備し、それを差し換えて表示させます。用紙単位でプロシージャに作成します。このプロシージャ名は、CRT**として、**の部分に連番を付けます。このプロシージャは、ここでは MainProgram にまとめました。ユーザが特定のパネルをクリックすると何をするかの案内メッセージを CRT**で表示させます。この番号は<**>と括って、コンソール画面の左上に表示させます。この番号は、プログラムが現在どの状態（ステータス）にあるかを識別するステータスコード(Istatus)として使い、そのコードに合わせて表示用プロシージャ CRT**が選ばれ、パネルのクリック処理も選択させます。基本的には、【戻る】はエスケープ、キャンセルなどのキー操作に相当し、【OK】は CR キーまたは Enter キーを打つ操作に替えるように画面を選択させます。

文字寸法は画面寸法に比例させて表示する

コンソールウインドウは、親ウインドウの枠の中で自由な位置と寸法を取ることができますが、案内表示の枠は縦横は 400：640 の固定比率です。その中で最大に接するように表示枠が取られ、その寸法に比例するように文字のフォントも変えます。例図は第１章の図１を見て下さい。ここには説明用の線図を描くこともできます。ウインドウの寸法を変えたり、また別のウインドウが重なって、それが外れたりするときは、再描画(AutoRedraw)が効くようにします。画面寸法が変るとシステムがイベントを発行し、コールバック関数の Paint が働きますので、そこで上記のステータスコードを判定して所定の画面を再描画させるコードを記述します。

メニュー項目は最小限に抑えたこと

コンソールウインドウがアクティブになったときに表示される標準のメニュー項目は、FORM, HELP です。このメニューが選択されたときのコールバック関数を、フォームモジュールのコード記述部分に書き込みます。FORM と HELP はプログラム全体で共通に利用することを考えて、そこから外部の標準モジュール CommonMenu1.bas に制御を渡して処理させます。このうち、Form_Unload イベントは、ウインドウの閉じるボタンをクリックすると呼ばれ、プログラム全体の終了を行わせます。なお、ウインドウの表示方法を変えるメニュー名は WINDOW またはウインドウではなく FORM にしてあります。グラフィックスのメニューに WINDOW を使いますので、それと区別するためです。FormConsole のメニューの詳細と、コード記述部分のリストを下に示します。

表 3.1　FormConsole のメニュー項目

メニュー	サブメニュー	説　明
FORM	Startup	総ての子ウインドウの、親ウインドウの中での配置を、プログラムの立上げ時（設計時）のレイアウトに並べ直します。
	Horizontal	子ウインドウを横長にして、親ウインドウの中で上下に並べます。このメニューが呼ばれる前にアクティブになっている子ウインドウが最上段にきます。
	Vertical	子ウインドウを縦長にして、親ウインドウの中で左右に並べます。このメニューが呼ばれる前にアクティブになっている子ウインドウが最左にきます。
	Cascade	アクティブになっている子ウインドウを大きくして重点的に見たいときに使い、他の子ウインドウは、裏に重ねた表示になります。
HELP	Version	日本の書物の終りには、奥付と言う書誌事項を記載する習慣があります。Version は、書物の場合の版番号に当たります。Version メニューは、プログラムの書誌事項を記載する AboutBox のメッセージを表示するものです。
	Help	主に、ユーザに対してプログラムの使い方を案内する目的持つものを Help と言います。この方法には種々ありますが、印刷物にするマニュアルを電子化したものと考えることができます。Prototype の Help サブメニューは、このプログラム専用にコンパイルした HTMLHelp を呼び出して表示します。

備考：　この二つのメニュー項目（FORM, HELP）は、総ての子ウインドウで共通に利用します。ただし、メニューの数は子ウインドウごとに違いますので、メニューバー上の位置は動きます。共通に利用するメニューですので、処理は標準モジュール CommonMenu の中の MenuWindow と MenuHelp のプロシージャを呼ぶようにしてあります。

```
Private Sub Form_Load()
    ScaleMode = 1
    aspect0 = 0.625
    CMonitor.FillColor = QBColor(15) '(White)
    CMonitor.FillStyle = 0
End Sub
'-------------------------------------------------------------------
Private Sub Form_Resize()
    If WindowState = 1 Then Exit Sub
    ConsolePaint
End Sub
'--------------------------- Text Input from Console Input Window
Private Sub KeyInText_KeyPress(KeyAscii As Integer)
    Dim KeyInputRecord$
    If KeyAscii = vbKeyReturn Then '====レターンキーが押されたとき
        KeyInputRecord$ = FormConsole.KeyInText.text
        Write6 KeyInputRecord$
        FormConsole.KeyInText.text = ""
    End If
End Sub
'-------------------------------------------------------------------
Private Sub mWind_Click(Index As Integer)
    MenuWINDOW Index
End Sub
'-------------------------------------------------------------------
'HELP Menu ======================================
Private Sub mHelp_Click(Index As Integer)
    MenuHelp Index
End Sub
'-------------------------------------------------------------------
Public Sub Form_Unload(Cancel As Integer)
    MenuUnload Cancel
End Sub
```

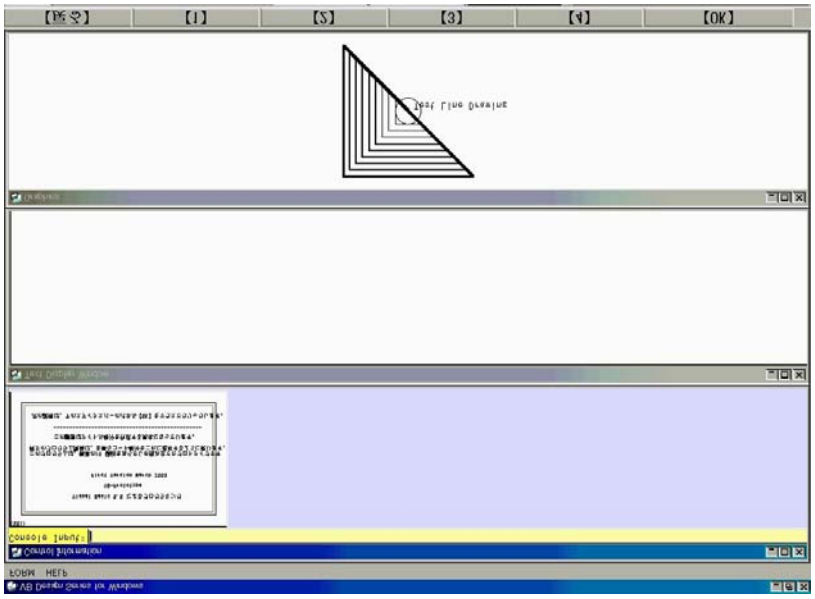

図 2.1　Horizontal のレイアウト

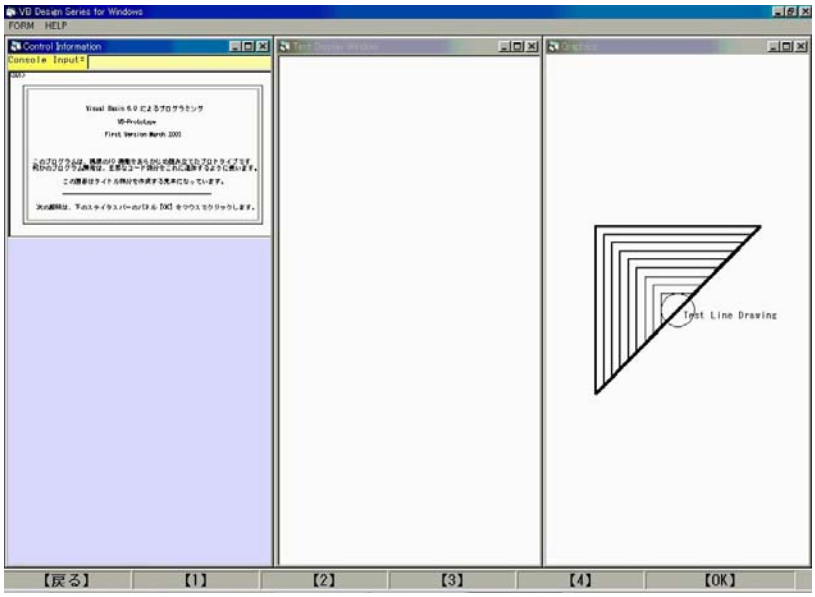

図 2.2　Vertical のレイアウト

図 2.3　Cascade のレイアウト

3.3 FormLprint

プリンタ用のバッファーとして使う

電卓（卓上電子計算器）を使う程度の計算であれば、表示画面の結果を手書きで紙に写すことでも実用になります。しかし、実務にコンピュータを利用するときは、モニタ上で計算結果を確認すると同時に、メモ的に途中経過をプリンタに出力させておきたいことがあります。また、レポートに使うような清書のプリントも必要です。DOS のモニタ上のリストは一過性の表示ですので、行数の多い結果の表示には向きません。N88Basic では、モニタに書き出すときは Print 文、プリンタへの出力は Lprint 文を使い分ける必要がありました。子ウインドウの FormLprint は、Lprint を意識した命名です。このウインドウには、スクロールの効くテキストエディタ用にリッチテキストボックスを載せ、ここをプリンタ出力のバッファーにします。これが FormLprint の考え方です。システムからの文字出力のモニタ表示が目的ですが、コンソールのテキスト入力枠からの入力文字のエコー表示にも使います。ビットマップ画像もクリップボードを介して貼付けることができます。そして、この全体をプリンタに出力させることができます。

標準入出力文との対応を付けること

N88Basic, Quick Basic の言語仕様には、コンソール入力に INPUT 文があり、テキストから自動的にフォーマット変換して変数に代入できました。逆に、変数からテキストへの変換には PRINT USING 文でフォーマット変換しました。プリンタへの書き出し専用には LPRINT 文がありました。ちなみに、Fortran 言語の標準入出力文は Read(5,…), Write(6, …)の形式であり、…の部分にフォーマット定義文のラベルが使用されます。ところが、VB の言語仕様では、入出力と連動する自動フォーマット変換は、ファイルからの入出力文の Input#、Write#、Print#だけです。LPRINT が無くなり、すべて Print 文が利用されます。それに伴って、テキストフォーマットの USING の仕様が使えなくなりました。そこで、内部データの書式変換をして文字列に直す処理を ConsoleIO のモジュールにまとめました。

フォーマット変換は面倒であること

数値変数を文字に直して書き出す際に使う VB の書式仕様は、Format 関数で行います。変数並びは、一旦 Format 関数で文字並びのデータに変換します。リッチテキストボックスへの書き出しのフォーマット変換にも応用します。リッチテキストボックスに文字列を書き出すとき、文字フォントは原則としてプロポーショナルフォントですので、文字数を勘定して複数行に渡る文字列を縦に揃えることができなくなりました。これはリストを表形式で印刷させてレポートの清書を作成したいときに困ります。変数の数値などの印刷には、書式仕様を決めてフォーマット変換をしますが、この変換と書き出しとには専用の関数を作成しました。この関数は、標準モジュール ConsoleIO0 にまとめました。リッチテキストボックスは、清書用のデータのプリントイメージを確認するだけでなく、種々の書き込みを受け入れるデータのバッファーの目的で利用します。この FormLprint 画面で簡単な編集ができます。データをそのままプリンタに送って印刷もできますが、質の良い編集をするには、このデータをファイルに保存し、WORD などを使って成果品にまとめるのがよいでしょう。

表 3.2　FormLprint のメニュー

メニュー	サブメニュー	説　明
FILE	SaveLprint	表示されているテキストをファイルに保存します。
	Print	プリンタへ出力します。
FONT	文字の寸法、字体、色を指定できます。	
FORM	FormConsole のメニューと同じですので、そちらで説明を見て下さい。	
HELP		
備考： EDIT メニューの定番は、select all/undo/cut/copy/paste です。これは、ショートカットキーを使う方法（ctrl+ A/Z/X/C/V）で処理できますので、メニューには含ませていません。		

```
Private Sub Form_Resize()
    If WindowState = 1 Then Exit Sub
    RichTextBox1.Top = 0
    RichTextBox1.Left = 0
    RichTextBox1.Width = FormLprint.Width
    RichTextBox1.Height = FormLprint.Height - 300
End Sub
'---------------------------------------------------------------------
Private Sub mFile_Click(Index As Integer)
    Select Case Index
        Case 1: SaveLprint
        Case 2: Lprinter
    End Select
End Sub
'---------------------------------------------------------------------
Private Sub mnuFont_Click(Index As Integer)
   MDIFormConsole.MDIDialog.Flags = cdlCFScreenFonts Or cdlCFEffects
    MDIFormConsole.MDIDialog.FontName = "ＭＳ ゴシック"
    MDIFormConsole.MDIDialog.FontSize = 10
    MDIFormConsole.MDIDialog.ShowFont
    RichTextBox1.SelColor = MDIFormConsole.MDIDialog.Color
    RichTextBox1.SelFontName = MDIFormConsole.MDIDialog.FontName
    RichTextBox1.SelFontSize = MDIFormConsole.MDIDialog.FontSize
    RichTextBox1.SelBold = MDIFormConsole.MDIDialog.FontBold
    RichTextBox1.SelItalic = MDIFormConsole.MDIDialog.FontItalic
    RichTextBox1.SelStrikeThru = MDIFormConsole.MDIDialog.FontStrikethru
    RichTextBox1.SelUnderline = MDIFormConsole.MDIDialog.FontUnderline
End Sub
'---------------------------------------------------------------------
Private Sub Lprinter()
    Dim intErrNum As Integer
    MDIFormConsole.MDIDialog.Flags = cdlPDPrintSetup
    With MDIFormConsole.MDIDialog
        .CancelError = True
        .Flags = cdlPDReturnDC + cdlPDUseDevModeCopies
    End With
    If RichTextBox1.SelLength = 0 Then
        MDIFormConsole.MDIDialog.Flags = MDIFormConsole.MDIDialog.Flags + cdlPDNoSelection
    Else
        MDIFormConsole.MDIDialog.Flags = MDIFormConsole.MDIDialog.Flags + cdlPDSelection
    End If
    On Error GoTo Errcancel
    MDIFormConsole.MDIDialog.ShowPrinter
    RichTextBox1.SelPrint MDIFormConsole.MDIDialog.hDC
    Exit Sub
Errcancel:
    Err.Clear
End Sub
```

なお、 Form_Unload()、mWind_Click()、mHelp_Click()のイベント処理プロシージャは、FormConsoleに示したものと同じですのでリストから省きました。

3.4 FormGraphic

文字出力も考えたグラフィックス専用とする

DOS のパソコンでは、キャラクタディスプレイとグラフィックスディスプレイとは表示原理が異なるのですが、この二つを同じモニタ画面を論理的に切り替えて表示していました。ここでの FormGraphic はグラフィックスディスプレイ専用とし、キャラクタディスプレイ用の FormConsole と機能を分離しました。Windows では、文字寸法が自由に選択できますが、デフォルトではプロポーショナルフォントを使いますので、例えば、製図で寸法線を引いて寸法文字を記入するなどのとき、文字と線図との相性が悪くなります。そのため、Prototype で使うのは、デフォルトで、すべて等幅フォントの MS ゴシックにしました。FormGraphic のウインドウは、作図命令で図を表示する目的だけに使い、ユーザがこの画面で何かの作業をすること、例えば座標読取りなどをしません。しかし、表示される画像全体の寸法を変えて表示させることや、色や線種を変えてレポートに貼付ける、などの手段を提供することにしました。これらの大部分はメニューの項目で処理します。基本的なメニューを表 3.3 に一覧を示します。選択肢が多くならないように、項目数は制限しました。グラフィックス処理は、これだけでも独立した幾つものプログラムができるほど多様なプロシージャがあります。これに関しては、アプリケーションプログラムとして別に開発した VB_Graphics があります。このプロトタイプでは、その中の基本機能を抜粋して簡単にしてあります。

画板をフォームに載せるという処理をする

FormGraphic の作業領域には、仮想の画板をフォームの作業域に載せると考え、そこに作図します。この画板は、DOS の標準モニタ画面をエミュレートすることを想定し、640×480 のアスペクト比（4：3）で中央に最大に接するように載せます。ここでの 640×480 の寸法数値は仮想の値です。グラフィックスの概念で言うと、画板に相当するビューポートですので、フォームのアスペクト比と画板のアスペクト比とが同じでないと、フォームの左右または天地に余分の作図領域を使うことができます。

FormGraphic の画像は、原理的には三段階の過程で表示されます。まず

- ①：表示させたい画像はカメラでフィルムに撮影すると考え、仮想のフィルムを想定します。このフィルムは横：縦の寸法比だけを考えます。デフォルトは標準テレビの画面アスペクト比と相似の 4：3（1：0.75）とします。これは 640×480 の標準モニタの画面を想定しています。ちなみに、N88Basic の画面では 1：0.625、35 ミリカメラでは 3：2（1：0.67）、JIS の用紙を横長に使うと 1：0.7、です。このアスペクト比は、メニューの WINDOW-Window で変更できます。
- ②：カメラは、被写体が存在する二次元の世界座標を上記のアスペクト比の矩形で切り取るように撮影します。このとき、カメラのレンズ中心が狙う二次元の世界座標（Wx, Wy）と、視野に納める横幅寸法（Ww）の三つの数値を決めます。視野の高さの方は、フィルムのアスペクト比から計算します。
- ③：このフィルム上の像を引き伸ばしてビューポートに焼きつけます。そのとき、ビューポート上に（Wx, Wy, Ww）を再現させる座標系を割り付けます。これがウインドウ・ビューポート変換です。このウインドウは、グラフィックスの方で言うウインドウであって、原義の窓の意味が強く、窓を通して見る外の世界の矩形寸法を指定します。Windows システムで言う通称のウインドウではありません。

フィルムの焼き付け方法が選択できること

ビューポートは実質的な作画面ですので、そこに固有の座標系があります。最も原始的な座標系は、左上を原点とし、ピクセル数で座標値を与えます。これでは数学的に使い難いので、ウインドウ・ビューポート変換でユーザの座標系を割り付けます。このとき、フィルムの向きを入れ替えて焼きつけると画像の向きが上下・左右入れ代わります。また、ビューポートの縦横比に合わせるようにフィルムを引き伸ばせば、元の画像とは違った縦長、または横長の図が得られます。この操作を WINDOW-Viewport メニューに集めます。なお、フィルムを任意の角度で回転させる焼きつけは含ませていません。

表 3.3　FormGraphic で定義するメニュー

メニュー	サブメニュー	サブサブメニュー	説　明
FILE	Print		プリンタへ画像出力
EDIT	ClearScreen	CLS	画面の消去
		PictureCLS	背景の画像を消去
	CopyToClipBrd		クリップボードへ画像のコピー
	SaveBmpFile		ファイルへビットマップ画像の保存
AUTOREDRAW	false/true		再描画のモード指定
WINDOW	Viewport	Isotropic	等方性画像の作画
		An-isotropic	フォームに合わせて伸び縮みする作画
	Rotate	Normal	数学座標での投影
		UpsideDown	Y 軸の正の向きは下向き
		LeftsideRight	左右逆の図形に作図
		TurnRound	180 度回転させて作図
	Window		グラフィックスウインドウの定義変更
LINE	Style	SolidLine	実線・破線・点線などの線種を選択できますが、これは下の細線(Thin)だけに効き、他の太さの線では全て実線で作図されます。
		DashedLine	
		DottedLine	
		DotDash	
		DotDotDash	
	Width	Thin	線の太さの変更は、実用的には 3 種類あれば充分です。
		Medium	
		Thick	
FONT			文字の寸法、字体、色を指定
FORM	FormConsole のメニューと同じですので、そちらで説明を見て下さい。		
HELP			

```
'-----------------------------------------------------------------
Option Explicit
Dim strReadFileName As String
Dim XLabel$, YLabel$, WLabel$, RLabel$
Dim wx0$, wy0$, WW0$, wr0$
Private Sub Form_Load()
    ScaleMode = 1
    FormGraphic.AutoRedraw = False
    FormGraphic.Top = 0
    FormGraphic.Left = 0
    Wwx = 0: Wwy = 0: Www = 640: Gaspect0 = 0.75
    Dpwind Wwx, Wwy, Www
    FormGraphic.Text1.text = "0"
    FormGraphic.Text2.text = "0"
    FormGraphic.Text3.text = "640"
    FormGraphic.Text4.text = "0.75"
    Isotropic = 1
    Iproc = 1
    Irotate = 1
    Frame1.Visible = False
    FormGraphic.FillColor = QBColor(15) '(White)
    FormGraphic.FillStyle = 0
    End Sub
'-----------------------------------------------------------------
Private Sub Form_Resize()
    If WindowState = 1 Then Exit Sub
```

```
        Dpwind Wwx, Wwy, Www
End Sub
'---------------------------------------------------------------------
Private Sub Form_Paint()
    GraphicPaint
End Sub
'---------------------------------------------------------------------
'FILE Menu ==========================================
Private Sub mFileInd_Click(Index As Integer)
    Dim IX, imode As Integer
    Select Case Index
        Case 3
            imode = FormGraphic.AutoRedraw
            If imode = False Then
                FormGraphic.AutoRedraw = True
                GraphicPaint
            End If
                MDIFormConsole.MDIDialog.Flags = cdlPDPrintSetup
                MDIFormConsole.MDIDialog.CancelError = True
                On Error GoTo Errcancel
                MDIFormConsole.MDIDialog.ShowPrinter
                PrintForm
                FormGraphic.AutoRedraw = imode
    End Select
    Exit Sub
Errcancel:
            Err.Clear
End Sub
'---------------------------------------------------------------------
'EDIT Menu ==========================================
Private Sub mEdit_Click(Index As Integer)
    Select Case Index
        Case 1
           'ClearScreen
        Case 2
            GraphicPaint
        Case 4: CopyToClipboard
        Case 8: SaveBitmapfile
    End Select
End Sub
'---------------------------------------------------------------------
'ClearScreen
Private Sub mCls_Click(Index As Integer)
    Select Case Index
        Case 1
            FormGraphic.Cls
        Case 2
           FormGraphic.Picture = LoadPicture()
    End Select
End Sub
'---------------------------------------------------------------------
Private Sub CopyToClipboard()
    Dim imode As Integer
            imode = FormGraphic.AutoRedraw
            If imode = False Then
                FormGraphic.AutoRedraw = True
                GraphicPaint
            End If
            Clipboard.Clear
            Clipboard.SetData FormGraphic.Image, vbCFBitmap
            FormGraphic.AutoRedraw = imode
End Sub
'---------------------------------------------------------------------
Private Sub SaveBitmapfile()
    Dim strSaveFileName As String
    Dim imode As Integer
    MDIFormConsole.MDIDialog.Filter = "Bitmap File(*.bmp)|*.bmp"
    MDIFormConsole.MDIDialog.ShowSave
```

```
    strSaveFileName = MDIFormConsole.MDIDialog.FileName
    If strSaveFileName <> "" Then
            imode = FormGraphic.AutoRedraw
            If imode = False Then
                FormGraphic.AutoRedraw = True
                GraphicPaint
            End If
            SavePicture FormGraphic.Image, strSaveFileName
            FormGraphic.AutoRedraw = imode
    End If
End Sub
'---------------------------------------------------------------------
'AUTOREDRAW Menu ====================================
Private Sub mAuto_Click(Index As Integer)
    Select Case Index
        Case 1
            FormGraphic.AutoRedraw = False
            mAuto(1).Checked = True
            mAuto(2).Checked = False
        Case 2
            FormGraphic.AutoRedraw = True
            mAuto(1).Checked = False
            mAuto(2).Checked = True
    End Select
End Sub
'---------------------------------------------------------------------
'WINDOW Menu ========================================
'Viewport
Private Sub mIso_Click(Index As Integer)
    Select Case Index
        Case 1
            Isotropic = 1
            mIso(1).Checked = True
            mIso(2).Checked = False
        Case 2
            Isotropic = 0
            mIso(1).Checked = False
            mIso(2).Checked = True
    End Select
    GraphicPaint
End Sub
'---------------------------------------------------------------------
'Rotate
Private Sub mRotate_Click(Index As Integer)
    Dim i, id As Integer
    id = Index
    If id = 0 Then id = 1
    For i = 1 To 4
        mRotate(i).Checked = False
    Next i
    Irotate = id
    mRotate(id).Checked = True
    GraphicPaint
End Sub
'---------------------------------------------------------------------
'Window
Private Sub mnWind_Click()
    Frame1.Visible = True
    XLabel$ = Text1.text
    YLabel$ = Text2.text
    WLabel$ = Text3.text
    RLabel$ = Text4.text
End Sub
'---------------------------------------------------------------------
'Default Button Click
Private Sub Command1_Click()
    Text1.text = "0"
    Text2.text = "0"
```

```
    Text3.text = "640"
    Text4.text = "0.75"
    wx0$ = Text1.text
    wy0$ = Text2.text
    WW0$ = Text3.text
    wr0$ = Text4.text
End Sub
'---------------------------------------------------------------------
'Cancel Button Click
Private Sub Command2_Click()
    Text1.text = XLabel$
    Text2.text = YLabel$
    Text3.text = WLabel$
    Text4.text = RLabel$
    Frame1.Visible = False
End Sub
'---------------------------------------------------------------------
'OK Button Click
Private Sub Command3_Click()
    Dim IX As Integer
    On Error GoTo 100
    wx0$ = Text1.text
    wy0$ = Text2.text
    WW0$ = Text3.text
    wr0$ = Text4.text
    Wwx = CSng(Text1.text)
    Wwy = CSng(Text2.text)
    Www = CSng(Text3.text)
    Gaspect0 = CSng(Text4.text)
    If Gaspect0 < 0.1 Then GoTo 100
    Frame1.Visible = False
    Exit Sub
100 IX = MsgBox("Invalid number", 0)
    Text1.text = XLabel$
    Text2.text = YLabel$
    Text3.text = WLabel$
    Text4.text = RLabel$
End Sub
'---------------------------------------------------------------------
'LINE Menu ======================================
'Line Style
Private Sub mLineStyle_Click(Index As Integer)
    Dim i, id As Integer
    id = Index
    If id = 0 Then id = 1
    For i = 1 To 5
        mLineStyle(i).Checked = False
    Next i
    Select Case id
        Case 1: FormGraphic.DrawStyle = 0
        Case 2: FormGraphic.DrawStyle = 1
        Case 3: FormGraphic.DrawStyle = 2
        Case 4: FormGraphic.DrawStyle = 3
        Case 5: FormGraphic.DrawStyle = 4
    End Select
    mLineStyle(id).Checked = True
    GraphicPaint
End Sub
'---------------------------------------------------------------------
'Line Width
Private Sub mLineW_Click(Index As Integer)
    Dim i, id As Integer
    id = Index
    If id = 0 Then id = 1
    For i = 1 To 4
        mLineW(i).Checked = False
    Next i
    Select Case id
```

```
        Case 1: FormGraphic.DrawWidth = 1
        Case 2: FormGraphic.DrawWidth = 2
        Case 3: FormGraphic.DrawWidth = 4
    End Select
    mLineW(id).Checked = True
    GraphicPaint
End Sub
'-----------------------------------------------------------------------
Private Sub mnuFont_Click()
    MDIFormConsole.MDIDialog.Flags = cdlCFScreenFonts Or cdlCFEffects
    MDIFormConsole.MDIDialog.FontName = "ＭＳ ゴシック"
    MDIFormConsole.MDIDialog.FontSize = 10
    MDIFormConsole.MDIDialog.ShowFont
    FormGraphic.ForeColor = MDIFormConsole.MDIDialog.Color
    FormGraphic.FontName = MDIFormConsole.MDIDialog.FontName
    FormGraphic.FontSize = MDIFormConsole.MDIDialog.FontSize
    FormGraphic.FontBold = MDIFormConsole.MDIDialog.FontBold
    FormGraphic.FontItalic = MDIFormConsole.MDIDialog.FontItalic
    FormGraphic.FontStrikethru = MDIFormConsole.MDIDialog.FontStrikethru
    FormGraphic.FontUnderline = MDIFormConsole.MDIDialog.FontUnderline
End Sub
'-----------------------------------------------------------------------
```

なお、 Form_Unload()、mWind_Click()、mHelp_Click()のイベント処理プロシージャは、FormConsole に示したものと同じですのでリストから省きました。

3.5 FormAboutBox

奥付に相当する情報を表示する

日本の書物では**奥付**があって、そこに著者・発行所・版数などを記す習慣があります。欧米の著作物にはこれと同等の定型化した習慣はありません。しかし名詞として colophon, imprint の訳語があります。プログラムも著作物ですので、奥付のような表記が必要になりますが、一般的にこの表記のためのウインドウを AboutBox（...について、の意）と言います。Windows のアプリケーションではバージョン情報(Version)を載せる一つのフォームに作ります。VB6 の開発ツールにもバージョン表示用の標準デザインが準備されていますが、ここでの FormAboutBox のように、プログラマレベルで標準モジュールを作成することができます。

HELP-Version メニューで参照する

AboutBox を参照するために、メニュー項目 HELP に Version のサブメニューを入れてあります。第 3.2 節の FormConsole のメニュー説明を参照して下さい。Macintosh の画面では、メニューバー最左端の Apple アイコンの箇所に AboutBox を開くメニューを載せるのが定型になっています。奥付の習慣は欧米にはありませんので、About とか Version に何を記述するかの定番は無いようです。奥付となれば、記述の中身が大体決まります。そのため、FormAboutBox のプロトタイプを準備することにしました。中身は、無論、目的に合った内容に書き換えて使います。AboutBox は、一種のメッセージボックスですので、このウインドウはモニタの最前面に表示させます。そして、この AboutBox が閉じられるまで、他のウインドウは非アクティブにします。

```
'---------------------------------------------------------------------
Private Sub cmdOK_Click()
    FormAboutBox.Visible = False
    MDIFormConsole.Enabled = True
End Sub
'---------------------------------------------------------------------
Private Sub Form_Paint()
    Cls
    FormAboutBox.Print "       Version 2003-10-24"
    FormAboutBox.Print "    Last Compiled : 2005-02-01"
    FormAboutBox.Print "   Designed and Programmed by"
    FormAboutBox.Print "           SHIMADA S."
End Sub
'---------------------------------------------------------------------
Private Sub Form_Unload(Cancel As Integer)
    Cancel = False
    cmdOK_Click
End Sub
'---------------------------------------------------------------------
```

4. 標準モジュールの解説

4.1 N88sub

N88Basic のソースコードを利用するアイディア

NEC の 16 ビットパソコン PC98 を利用して N88Basic で開発したプログラムを VB6 に移植する際、作業を便利にするため、SCREEN 文・LOCATE 文の二つを、Nscreen・ Nlocate に書き換えれば済むようにしました。次ページに示すリストは VB6 コードを示したもので、プロトタイプの標準モジュール N88sub1 に作成しました。一般に、DOS の環境で開発されたプログラムでは、モニタへの文字表示は等幅フォントを使い、スクリーンへの文字詰めも半角で 80 文字×30 行（PC98 では 25 行）と一定でした。これに合わせて文字表示の書式や体裁がデザインされていました。Windows の環境では、フォーム（ウインドウ）の寸法もアスペクト比も変えることができます。しかし、文字のポイント寸法は連続的に変更することができませんし、文字幅もプロポーショナルフォントが標準になりましたので、プログラム側からユーザへ伝える文字情報の書式デザインが面倒になりました。テキストボックスでは、フォームの幅に合わせて自動改行などの機能を使うことができますが、単純にフォームに Print 文で文字を書き出すと、スクロール機能がないので、右端や下端が欠ける危険があります。

コンソールをメッセージボックスとして使う

Windows の環境で文字を出力させる方法は、「①ある一定の文字詰めの書式を考え、それが納まるようにフォームの寸法の方を固定にする」または「②その書式がフォームの寸法に対応して拡大または縮小して表示させる」を使い分けます。①の固定デザインは、メッセージボックスや、前章で解説した FormAboutBox のフォームがそうです。ここで解説する N88sub1 は、②を目的としたものです。この書式は、N88Basic で採用されていた 640×400 の固定アスペクト比を持ったピクチャーボックスを FormConsole のフォームに左上詰めで一杯に接する様に載せます。この寸法に合わせてウインドウ・ビューポート変換の処理をします。これがプロシージャ名の Nscreen です。

コンソールには二種類の座標系がある

文字の出力は Print 文です。線の描画は Line 文で行うことができますので、簡単な説明用の線図も同じ画面に描くことができます。立上げ時のコンソールには、線の太さを変えた二つの矩形枠を描いて、その中に納まるように文字を書いた例です。N88Basic では、文字書き出し位置の座標は**キャラクタ座標**が用いられ、線の作図に使う座標は**ピクセル座標**が使われました。キャラクタ座標の指定が LOCATE 文です。この座標系の定義方式を VB6 でそのまま生かすようにするプロシージャ名を Nlocate としました。したがって、N88Basic のコンソール表示部分は、このプロシージャ名に置き換えることと、Print 文、Line 文に FormConsole.の修飾子を付けることで同質のコンソール表示が得られます。文字フォントは、実際の表示寸法を元に、80 文字×25 行に納まるポイント数を割り出して表示に使います。使用する文字フォントは、等幅の MS ゴシックです。ポイント数は連続的に選べませんので、フォームの寸法変化に滑らかに対応できません。特に半角の英数字幅の二倍と全角文字と対応が微妙にズレます。

文字書き出しは二種類の座標系を使い分ける

コンソールに文字を表示する VB6 のメカニズムは、DOS の環境での Quick Basic の仕様を引き継いでいて、原理的に二種類あります。この種類を**キャラクタディスプレイモード**と、**グラフィックスディスプレイモード**と区別します。前者では、Print 文は改行込みの命令であって、一つの Print 文が済むと、次ぎの書き出し位置が文字ポイント数に合わせて自動的に一行分下に移ります。後者は、Nlocate 文を先行させた Print 文ですが、これはその場所限りの位置指定です。続けて Print 文で書き出すときは、キャラクタディスプレイモードになって、コンソールの最下行相当の位置に書き出します。このため、文字列の行並びの設計では、Nlocate 文を使って行位置を指定する Print 文と、それ無しの場合とを混用すると思った通りの書式になりません。また、行間隔も微妙に異なります。コンソール出力の例題は、プロトタイププロジェクトの二つの例題プロシージャ CRT01, CRT02 を比較して見て下さい。

```
Option Explicit
Public aspect0 As Double            '--------- graphics window constants
'=================================================
'       N88-Console Screen-Control Subroutines
'=================================================
'-------------------------------------------------------------------------
Public Sub Nscreen()
    Dim asp0 As Double, pwidth As Double
    Dim pheight As Double, ppwidth As Double, ppheight As Double
    Dim Jfont As Integer
    Dim Yheight As Double
'
    FormConsole.ScaleMode = 1
    pwidth = FormConsole.Width
    pheight = FormConsole.Height - 2.5 * FormConsole.Prompt.Height
    asp0 = pheight / pwidth
    If asp0 > aspect0 Then
        ppwidth = pwidth
        ppheight = pwidth * aspect0
    Else
        ppwidth = pheight / aspect0
        ppheight = pheight
    End If
'
    FormConsole.CMonitor.Width = ppwidth
    FormConsole.CMonitor.Height = ppheight
    FormConsole.CMonitor.Top = FormConsole.Prompt.Height
    FormConsole.CMonitor.Left = 0
    Yheight = FormConsole.CMonitor.Height / Screen.TwipsPerPixelY
    Jfont = Yheight / 36
    FormConsole.CMonitor.FontSize = Jfont
    FormConsole.KeyInText.Width = FormConsole.Width - FormConsole.Prompt.Width

    FormConsole.CMonitor.ScaleLeft = 0
    FormConsole.CMonitor.ScaleTop = 0
    FormConsole.CMonitor.ScaleWidth = 640
    FormConsole.CMonitor.ScaleHeight = 400
    FormConsole.Show FormConsole.CMonitor
End Sub
'-------------------------------------------------------------------------
Public Sub Nlocate(X As Variant, Y As Variant)
    FormConsole.CMonitor.CurrentX = 8 * X
    FormConsole.CMonitor.CurrentY = 16 * Y
End Sub
'-------------------------------------------------------------------------
```

4.2 VBGsub

作図用デバイスドライバの考え方を使うこと

プロトタイプとして準備した標準モジュール VBGsub は、Dp**** の名前のグラフィックス用プロシージャを集めたものです。これは、プログラマレベルでの作図コマンドであって、対象画面は、FormGraphic です。コンソールの目的に使う FormConsole とグラフィックス作画に使う FormGraphic とは、どちらも文字と線図とを表示するウインドウです。どちらも、最終的なメソッド（サブルーチン）に Print 文と Line 文とを使います。しかし、二つのフォームの大きな違いは、座標系の考え方が違うことです。グラフィックス座標は基本的に数学座標系を使い、y 軸は上向きを正に取ります。文字並びを表示させるとき、文字位置の指定は文字列の左下隅を起点にします。これに対応させるため、FormGraphic への作図には、デバイスドライバの機能を持たせた Dp**** の名前のグラフィックス用プロシージャを介して表示させます。例えば、線の作図には Dpmove/Dpdraw の対のプロシージャを使い、文字の表示には、Dptext を使います。これは、書き出す文字のポイント数を調べ、文字の起点を文字の左下にするように位置を調整して書かせるプロシージャです。

ビューポートの座標系を設定すること

モジュール VBGsub にあるプロシージャの中でとりわけ重要なものは Dpwind です。これはグラフィックで言うウインドウ・ビューポート変換を定義するものであって、WINDOW-Window メニューで設定を変更できます。FormGraphic のメニューにある WINDOW は、グラフィックスの意義で使いますので、フォームの寸法変換に使うメニュー項目は FORM としてあります。この変換では、幾つかのコモン変数を使って画面の座標系を定義します。

- Wwx, Wwy, Www, Gaspect0 ：グラフィックスウインドウの定義です。デフォルトは（0、0、640、0.75）にしました。これは、横縦のアスペクト比（640×480）の画面で、画面中心を原点（0、0）とします。

- Irotate：座標軸の向きの正負を指定します。Irotate=1 は標準のデカルト座標であって、y 軸は上向きが正です。Irotate=2 は、下向きを正にする定義です。この変更は、MAPPING-Rotate メニューで設定できます。したがって、コンソールウインドウと同じキャラクタディスプレイモードに設定することもできます。このときは、上の Wwx, Wwy, Www, Gaspect0 の設定を（320、200、640、0.625）とし、Irotate=2 にして Dpwind を呼びます。

- Isotropic：FormGraphic のフォーム寸法は可変ですが、そこに考える仮想の画板は、アスペクト比が Gaspect0 で固定されています。これは図形の大小に関わらず、図形を相似に表示させるためです。この作画モードを等方性(Isotropic)と言います。そのため、仮想画板の左右または天地に余白が残ります。非等方性(An-isotropic)に指定すると、フォーム一杯に仮想の画板が広がります。そうすると、円は楕円で描かれます。この変更は、WINDOW-Viewport メニューで設定します。

```
'-------------------------------------------------------------------------
Option Explicit
Public Wwx As Double, Wwy As Double         '------ size of Window
Public Www As Double, GaspectO As Double    '--------- window constants
Public Isotropic As Integer
Public Irotate As Integer
Public ICountSecond As Integer
Public Icolor As Long
Public Const pie = 3.14159
Public Iproc As Integer
Dim asp As Double
'=================================================
'       Graphics Subroutines for Device Driver
'=================================================
'-------------------------------------------------------------------------
Public Sub Dpwind(xxO As Double, yyO As Double, WWO As Double)
    Dim pwidth As Double, pheight As Double         'size of form
    Dim xx As Double, yy As Double, ww As Double, wh As Double
'
    FormGraphic.ScaleMode = 1   'twip mode for window size
    xx = xxO: yy = yyO: ww = WWO: wh = ww * GaspectO
    pwidth = FormGraphic.Width
    pheight = FormGraphic.Height - MDIFormConsole.StatusBar1.Height   'size of effective height
    asp = pheight / pwidth
'
    If Isotropic = 1 Then
        If asp > GaspectO Then
            wh = ww * asp
        Else
            ww = wh / asp
        End If
    End If
'
    FormGraphic.ScaleMode = 0
    Select Case Irotate
        Case 1 'Normal
            FormGraphic.ScaleWidth = ww
            FormGraphic.ScaleHeight = -wh
            FormGraphic.ScaleLeft = xx - ww / 2
            FormGraphic.ScaleTop = yy + wh / 2
        Case 2 'BottomUp
            FormGraphic.ScaleWidth = ww
            FormGraphic.ScaleHeight = wh
            FormGraphic.ScaleLeft = xx - ww / 2
            FormGraphic.ScaleTop = yy - wh / 2
        Case 3 'LeftToRight
            FormGraphic.ScaleWidth = -ww
            FormGraphic.ScaleHeight = -wh
            FormGraphic.ScaleLeft = xx + ww / 2
            FormGraphic.ScaleTop = yy + wh / 2
        Case 4 'TurnOver
            FormGraphic.ScaleWidth = -ww
            FormGraphic.ScaleHeight = wh
            FormGraphic.ScaleLeft = xx + ww / 2
            FormGraphic.ScaleTop = yy - wh / 2
    End Select
'
End Sub
'-------------------------------------------------------------------------
```

```
Public Sub Dpmove(XP As Double, YP As Double)
    FormGraphic.CurrentX = XP
    FormGraphic.CurrentY = YP
End Sub
'-----------------------------------------------------------------------
Public Sub Dpdraw(XP As Double, YP As Double)
    Dim XQ As Single, YQ As Single
    Dim XO As Single, YO As Single
    XQ = XP
    YQ = YP
    XO = FormGraphic.CurrentX:
    YO = FormGraphic.CurrentY
    FormGraphic.Line (XO, YO)-(XQ, YQ), Icolor
End Sub
'-----------------------------------------------------------------------
Public Sub Dpensz(Ipsz As Integer)
    FormGraphic.DrawWidth = Ipsz
End Sub
'-----------------------------------------------------------------------
Public Sub Dperas()
    FormGraphic.Cls
End Sub
'-----------------------------------------------------------------------
Public Sub Dpcirc(xx As Double, yy As Double, rr As Double)
    Dim aspect As Single
    Dim XO As Single, YO As Single, rO As Single
    XO = xx: YO = yy: rO = rr
    aspect = asp
    If Isotropic = 1 Then aspect = 1
    FormGraphic.Circle (XO, YO), rO, Icolor, , , aspect
End Sub
'-----------------------------------------------------------------------
Public Sub Dpoint(xx As Double, yy As Double, imark As Integer)
    Dim XO As Single, YO As Single
    XO = xx: YO = yy
    If (imark = 0) And (FormGraphic.DrawWidth = 1) Then
        FormGraphic.PSet (XO, YO), Icolor
    Else
        Dpmove xx, yy
        Dpdraw xx, yy
    End If
End Sub
'-----------------------------------------------------------------------
Public Sub Dptext(xx As Double, yy As Double, ByVal text As String)
    FormGraphic.CurrentX = xx
    FormGraphic.CurrentY = yy
    FormGraphic.Print text
End Sub
'-----------------------------------------------------------------------
```

4.3 ConsoleIO

キーボードからの入力を受け付ける設計

コンピュータの文字データの入出力(I/O; input/output)は、コンソールタイプライタを使うことから出発しました。そのため、この作業と同質の処理を**標準入出力**とか、コンソール処理と言います。この項の表題に付けた ConsoleIO もこれを承けたものです。まず、文字の入力ですが、コンソールウインドウ FormConsole にはキーボードからの入力を受け付ける一つのテキストボックスを常駐させてあります。ここに入力されたテキストデータは、改行によってプログラムに取り込むことができますが、例題の Prototype ではその**エコー**を FormLprint のテキストウインドウに書き出すだけにしてあります。

書式変換が面倒であること

一般に、変数並びにデータを入力させるプログラミングコードは、書式なしの入力文が使われます。文字位置（カラム位置）を厳格に指定してデータリストを作成するのは、古い時代には行われていましたが、実作業に不便ですので、現在では書式なし入力文が普通になりました。データの**区切り記号**（**デリミタ**）にコンマ、タブ、スペース、改行コードなどが混用されています。例えば、Fortran では[READ(5,*) X,Y,Z]のようにスペースとコンマで区切り、N88Basic や Quick Basic では[INPUT X,Y,Z]のように書きます。これを承けるデータリストは、キーボードから[1.3、2.5、10.2]のようにキーインすれば、変数に数値が代入されます。しかし、VB6 ではファイルからの入力文 Input#は残りましたが、単純な Input 文の仕様がありませんので、これと同じ処理を行わせる処理が面倒になりました。一つの解決方法は、リストを一旦作業用テキストファイルに書き出し、これを Input#文で解読します。この例題は別のプログラムで紹介します。

文字出力はテキストウインドウを使う

コンソールへ出力は、アプリケーション側からユーザに種々の情報を伝えます。表示面積は広くありませんので、行数の多い結果は、プリンタの方に出力させるか、一旦ファイルに記録して、後処理でプリンタに落とします。プロトタイプの FormLprint のテキストウインドウは、このプリンタ用のモニタ兼バッファーです。ここに文字列を書き出す命令が **Write6** です。テキストウインドウはテキストエディタの機能がありますので、この画面で簡単な編集をして、ファイルへの保存、プリンタへの出力ができます。書式を整える処理は、数値の場合には有効数字を決め、小数点位置を揃えたり、スペースを適度に按分して左詰めまたは右詰めに揃えたり、などのことを指します。この目的には、数値データを Format 変換してテキスト形式に直す関数として **Nformat** を準備しました。

```
'-----------------------------------------------------------------------
Option Explicit
Public Sub SaveLprint()   ' Lprint のファイルへの保存
    Dim PrintF$, FNOUT$
    MDIFormConsole.MDIDialog.Filter = "RichTextFile(*.rtf)|*.rtf"
    MDIFormConsole.MDIDialog.ShowSave
    FNOUT$ = MDIFormConsole.MDIDialog.FileName
    If FNOUT$ <> "" Then
        PrintF$ = FNOUT$
        FormLprint.RichTextBox1.SaveFile PrintF$, rtfText
    End If
End Sub
'-----------------------------------------------------------------------
Public Sub Write6(strText As String)
    FormLprint.RichTextBox1.SelStart = LenB(FormLprint.RichTextBox1.text)
    FormLprint.RichTextBox1.SelText = strText & Chr$(10)
End Sub
'-----------------------------------------------------------------------
Public Function NFormat(Dvalue As Variant, Fmt$) As String
'format transformation for numerical data such as Nformat(3.14,"#####.####")
'
    Dim ibstrt As Integer, ibend As Integer
    Dim Ibnr As Integer, ippos As Integer
    Dim FmtA$, FmtAA$, FmtB$, FmtC$
```

```
    On Error Resume Next
    ibstrt = InStr(Fmt$, "#")               '#の始まる文字位置
    ibend = InStrRev(Fmt$, "#")             '#の終わる文字位置
    Ibnr = ibend - ibstrt + 1               'Nformat"####.##"の文字数
    FmtA$ = Mid(Fmt$, ibstrt, Ibnr)         'format 文字列の取り出し
    ippos = InStr(FmtA$, ".") - 1           '小数点位置の探索
    If ippos < 1 Then ippos = ibend         '小数点が無ければ最後尾
    FmtB$ = Replace(FmtA$, "#", "0", ippos) '#の文字列を"0"に置き換え
    FmtC$ = String(Ibnr, "@")               '右詰めにする format
    FmtAA$ = Format$(Dvalue, FmtB$)         '数値の format 変換
    FmtAA$ = Format$(FmtAA$, FmtC$)         '右揃えに変換
    NFormat = Replace(Fmt$, FmtA$, FmtAA$)  '変換した結果を埋め込む
End Function
'-------------------------------------------------------------------------
Public Function SFormat(Dvalue As Variant, Fmt$) As String
'format transformation for numerical data into scientific style
'
    Dim Ibnr As Integer
    Dim FmtA$, FmtAA$, FmtC$
    On Error Resume Next
    Ibnr = Len(Fmt$) + 2                    'Sformat"#.00000"の文字数
    FmtC$ = String(Ibnr, "@")               '右詰めにする format
    FmtAA$ = Format$(Dvalue, Fmt$)          '数値の format 変換
    SFormat = Format$(FmtAA$, FmtC$)        '変換した結果を埋め込む
End Function
'-------------------------------------------------------------------------
Public Function LFormat(VarData As Variant, Ibnr As Integer) As String
'format transformation for string data such as LFormat("string data, 15)
'
    Dim FmtC As String, Strdata As String
    FmtC = String(Ibnr, "@") & "!"          '左詰めにする format 文字列
    Strdata = CStr(VarData)
    LFormat = Format$(Strdata, FmtC)
End Function
```

4.4 CommonMenu

共通するメニュー処理を集めた

メニューバーは親ウインドウに表示されますが、それは現在アクティブになっている子ウインドウに従属するように設計されたメニューバーを転用します。メニューの項目のうち、FORM と HELP は、子ウインドウすべて共通の項目ですし、子フォームの閉じるボタンもすべて全体プログラムの終了に使います。そのため、子フォームのイベントプロシージャから、この CommonMenu に制御を移して、ここに共通メニュー処理をまとめました。メニューの HELP は、コンパイルされた HtmlHelp を呼ぶようにしました。バージョン表示は、一つの独立したウインドウを使って表示させます。これはメッセージボックス並みに動作させるようにして、バージョンウインドウが表示されている間は、親ウインドウ全体を非アクティブにします。

```
'------------------------------------------------------------------------
Option Explicit
Private Declare Function HtmlHelp Lib "HHCtrl.ocx" Alias "HtmlHelpA" _
   (ByVal hwndCaller As Long, ByVal pszFile As String, _
   ByVal uCommand As Long, dwData As Any) As Long
Private Const HH_DISPLAY_TOPIC = &H0
'------------------------------------------------------------------------
Public Sub MenuUnload(Cancel As Integer)
    Dim IX As Integer, iXX As Integer
    On Error GoTo 0
'
    IX = MsgBox("Ok or Cancel?", 1, _
                "Stopping Program ")
    If IX = 1 Then
        'Close   'if no files open then may cause fatal error.
        End
    End If
    Cancel = True
End Sub
'------------------------------------------------------------------------
Public Sub MenuHelp(Index As Integer)
    Dim Helpfile2$
    Select Case Index
        Case 1
            MDIFormConsole.Enabled = False
            FormAboutBox.Visible = True
        Case 2
            If HelpFile$ <> HtmlHelpFilterName$ Then
                MDIFormConsole.MDIDialog.Filter = HtmlHelpFilterName$
                MDIFormConsole.MDIDialog.ShowOpen
                HelpFile$ = MDIFormConsole.MDIDialog.FileName
            Else
                Exit Sub
            End If
            Call HtmlHelp(FormGraphic.hWnd, HelpFile$, HH_DISPLAY_TOPIC, ByVal 0)
    End Select
End Sub
'------------------------------------------------------------------------
Public Sub MenuWINDOW(Index As Integer)
    Select Case Index
        Case 1: MDIFormConsole.Arrange vbTileHorizontal
        Case 2: MDIFormConsole.Arrange vbTileVertical
        Case 3: MDIFormConsole.Arrange vbCascade
     End Select
End Sub
'------------------------------------------------------------------------
```

4.5 MainProgram

アプリケーションの入口を意識すること

このモジュールは、プログラマがコードを記述する一つのモデルとして作成しました。何かのプログラミングをするとき、システムが最初に入る入口を Main とするのが定番ですし、一種のキーワードにもなっています。意識的にプログラムの入口を指定しなくても済む場合も多いのですが、その場合でもシステム側はデフォルトの Main を内部で準備するようです。この部分の詳細はマイクロソフト社のブラックボックスです。入り組んだプログラムを開発するときは、プログラマレベルで Main プロシージャを明示的に設けるべきです。実行形式の Prototype プログラムにシステムから最初に入る入口（スタートアップ）は、親ウインドウ MDIFormConsole にしてあって、制御はそこの FormLoad に入ります。そこからすぐに、Main を呼びようにコーディングしました。

Main が行う主な処理

Main では、当該のプログラムに使用する幾つかの子フォームを Load 命令で呼び出し、それを親ウインドウ上に表示させます。この際、プログラム次第では、表１ に示したすべての子ウインドウを使う必要はありません。子ウインドウが Load されるとき、処理は子ウインドウごとのイベント処理プロシージャ Form_Load を一回だけ呼びます。ここに、その子ウインドウに必要な初期処理を記述します。続けて、種々のデータの初期設定プログラム（例えば Init）を処理した後に、コンソールウインドウにタイトル画面を（例えば CRT01 で）表示してイベント待ちに移行します。

案内表示方法の設計

ユーザが個人的に使うことを目的に組んだプログラムを別とすれば、実行形式のプログラムを使うための説明がどこかに必要です。それには下に示すような方法が使われています。

- ①ReadMe.txt のようなテキストファイルに説明をまとめます。テキスト形式ですと、大抵のシステムで読むことができます。ただし文書量は 32KB、または多くても 64KB の制限があります。
- ②少し分量が嵩む説明は、Word, HTML, PDF などの文書ファイルにまとめます。この場合には、この文書形式が読めるソフトウェアが必要になりますが、分量の制限はありませんし、またグラフィックスなどを利用できます。マニュアルとしてハードコピー化したものを電子化ファイルで提供することが増えています。
- ③プログラムの実行時に参照できる HELP を組み込みます。これを、上の二つに対して online help と言います。例えば、CUI の環境ではキーボードから HELP ***のように入力するか、***？ように入力するのですが、使い方が分らないとお手上げになります。その点では、GUI の環境での HELP メニューの使い方は分り易くなります。ただし、プログラミングは厄介です。そして、
- ④プログラムの実行時に適切な案内メッセージをモニタに表示させます。

上記の四種類の方法は組み合わせて利用されます。ここでは、④の方法を採用しています。

ウイザード方式の表示と制御を採用

例題として示したコンソール案内表示用のプロシージャは、CRT01, CRT02, CRT999 です。この名前の付け方は、表示画面を紙芝居のように多く準備することを考えて CRT**の名前で揃えました。したがって、同名のプロシージャがプロジェクト毎に使われています。幾つかの表示用のプロシージャを適切に使うため、表示単位ごとに内部的に使うステイタス番号 Istatus を割り振ってあります。ウインドウの寸法が変わったりして描き直しのイベントが発生したら、ConsolePaint のプロシージャに制御が入ります。画面のウインドウ・ビューポート変換のプロシージャ Nscreen を呼んだ後で、Istatus 番号に当たる表示プロシージャを選んで再描画をさせます。つまり、このプロシージャに、利用する全部の CRT**のプロシージャ名が使われます。例題の表示プロシージャ CRT01 と CRT02 は、表示の原理が違います。CRT01 では、グラフィックスディスプレイのモードで作成してあって、Line 文で枠を描き、文字書き出し位置を Nlocate 文で指定しています。CRT02 ではキャラクタディスプレイモードでの表示です。文字は Print 文の並びで書いてあります。行の字詰めと行数を見るために文字尺度が参考のために入れてあります。プログラムがどのように流れるかを制御するユーザインタフェースは、ステイタスバーのパネルをクリックすることで行います。この見本がプロシージャの PanelClick に例示しました。なお、グラフィックスは単純なテストパターンを描き出すだけの例題を示してあります。実行時にメニューを使ってグラフィックス出力の変化を確かめて下さい。

```
'--------------------------------------------------------------------------
Option Explicit
Public Istatus As Integer
Public FNOUT$, InputF$, PrintF$, HelpFile$
Public HtmlHelpFilterName$
'--------------------------------------------------------------------------
Public Sub Main()
    Load FormGraphic
    Load FormLprint
    Load FormConsole
    MDIFormConsole.Arrange vbTileVertical
    Init
    CRT01
End Sub
'--------------------------------------------------------------------------
Public Sub Init()
    HtmlHelpFilterName$ = "Prototype.chm"
End Sub
'--------------------------------------------------------------------------
Public Sub GraphicPaint()
    Dpwind Wwx, Wwy, Www
    Select Case Iproc
        Case 1: Proc1  'Dyagonal demonstration
        Case 2:
        Case 3:
        Case 4:
        Case 5:
    End Select
End Sub
'--------------------------------------------------------------------------
Public Sub ConsolePaint()
    Nscreen
    Select Case Istatus
        Case 1: CRT01
        Case 2: CRT02
        Case 3:
        Case 4:
        Case 999: CRT999
    End Select
End Sub
'--------------------------------------------------------------------------
Public Sub CRT01()
    Istatus = 1
    FormConsole.CMonitor.Cls
    FormConsole.CMonitor.Line (0, 0)-(639, 399), 0, B 'F
    FormConsole.CMonitor.Line (32, 32)-(607, 368), 0, B 'F
    FormConsole.CMonitor.Line (40, 40)-(600, 43), QBColor(7), BF
    FormConsole.CMonitor.Line (40, 43)-(43, 357), QBColor(7), BF
    FormConsole.CMonitor.Line (597, 43)-(600, 357), QBColor(7), BF
    FormConsole.CMonitor.Line (40, 357)-(600, 360), QBColor(7), BF
    Nlocate 0, 0: FormConsole.CMonitor.Print "<S01>"
    Nlocate 23, 5: FormConsole.CMonitor.Print "Visual Basic 6.0 によるプログラミング"
    Nlocate 23, 7: FormConsole.CMonitor.Print "          VB-Prototype"
    Nlocate 23, 9: FormConsole.CMonitor.Print "      First Version March 2003"
    Nlocate 7, 13: FormConsole.CMonitor.Print "このプログラムは、標準の IO 機能をあらかじめ組み立てたプロトタイプです"
    Nlocate 7, 14: FormConsole.CMonitor.Print "何かのプログラム開発は、主要なコード部分をこれに追加するように使います。"
    Nlocate 16, 16: FormConsole.CMonitor.Print "この画面はタイトル部分を作成する見本になっています。"
    Nlocate 16, 18: FormConsole.CMonitor.Print "----------------------------------------"
    Nlocate 8, 20: FormConsole.CMonitor.Print "次の説明は、下のステイタスバーのパネル【OK】をマウスでクリックします。"
End Sub
'--------------------------------------------------------------------------
Public Sub CRT02()
    Istatus = 2
    FormConsole.CMonitor.Cls
    FormConsole.CMonitor.Print "(1)....*....1....*....2....*....3....*....4....*....5....*....6....*....7....*....8"
    FormConsole.CMonitor.Print "(2)全角文字文字幅尺度１画画画画＊面面面面２画画画画＊面面面面３画画画画＊面面面面４"
    FormConsole.CMonitor.Print "(3)<S02>"
```

```
    FormConsole.CMonitor.Print "(4)                          ■     解     説     ■"
    FormConsole.CMonitor.Print "(5)"
    FormConsole.CMonitor.Print "(6)このプログラムは、Visual Basic を使ってグラフィックスのプログラミングをする"
    FormConsole.CMonitor.Print "(7)ための教育用に作りました。N88Basic, Quick Basic, Fortran を使って作成した"
    FormConsole.CMonitor.Print "(8)プログラムを、VB に移植するときのプロトタイプまたはテンプレートにします。"
    FormConsole.CMonitor.Print "(9)"
    FormConsole.CMonitor.Print "(10)ウインドウの寸法を変えると、それに合わせて文字寸法も変えて表示されます。ただし"
    FormConsole.CMonitor.Print "(11)文字幅はプロポーショナルですので、文字の縦の並びを整えることが難しくなります。"
    FormConsole.CMonitor.Print "(12)半角 80 文字×25 行の表示用に画面の上と左とに文字数を勘定する尺度を並べました。"
    FormConsole.CMonitor.Print "(13)FONT メニューで文字フォントを変更できますが、文字並びはその都度変わります"
    FormConsole.CMonitor.Print "(14)  "
    FormConsole.CMonitor.Print "(15)前ページの表示画面は、枠線を付けたグラフィックスディスプレイ形式の表示見本です。"
    FormConsole.CMonitor.Print "(16)Line 文で枠線を描き、文字書き出し位置は Nlocate 文で指定します。これに対して"
    FormConsole.CMonitor.Print "(17)このページはキャラクタディスプレイ形式での表示見本として作成しました。"
    FormConsole.CMonitor.Print "(18)"
    FormConsole.CMonitor.Print "(19)ウインドウの右上の閉じるボタンは、すべて、全体のプログラムの終了として機能します。"
    FormConsole.CMonitor.Print "(20)-----------------------------"
    FormConsole.CMonitor.Print "(21)グラフィックスを再描画します。                    【1】をマウスでクリックします。"
    FormConsole.CMonitor.Print "(22)前の画面に戻るときは、下のステイタスバーのパネル  【戻る】をマウスでクリックします。"
    FormConsole.CMonitor.Print "(23)次の説明を見るときは、下のステイタスバーのパネル  【OK】をマウスでクリックします。"
    FormConsole.CMonitor.Print "(24)  "
    FormConsole.CMonitor.Print "(25)  "
End Sub
'---------------------------------------------------------------------
Public Sub CRT999()
    Istatus = 999
    FormConsole.CMonitor.Cls
    FormConsole.CMonitor.Print "<S999>     "
    FormConsole.CMonitor.Print "■ ＊  処理の終了   ＊ ■"
    FormConsole.CMonitor.Print
    FormConsole.CMonitor.Print "プログラムの終了は閉じるボタンで行います"
    FormConsole.CMonitor.Print "-----------------------------------"
    FormConsole.CMonitor.Print "前の画面に戻ります。   --> 【戻る】"
    FormConsole.CMonitor.Print "最初の画面に戻ります。 --> 【OK】"
End Sub
'---------------------------------------------------------------------
Public Sub PanelClick(Panel As Variant)
    Dim icase As Integer, IP As Integer
    Select Case Istatus
        Case 1  '----CRT01
            icase = 1   'default selection for OK
            Select Case Panel
                Case "【OK】": CRT02
            End Select
        Case 2  '----CRT02
            Select Case Panel
                Case "【戻る】": CRT01
                Case "【1】": GraphicPaint
                Case "【2】":
                Case "【OK】": CRT999   'this sample enters into end procedure
            End Select
        Case 3  '----CRT03 this is an example for adding other procedures
            Select Case Panel
                Case "【戻る】": CRT02
                Case "【1】":
                Case "【2】":
                Case "【OK】": 'CRT04
            End Select
        Case 999  '---- CRT999
            Select Case Panel
                Case "【戻る】": CRT02
                Case "【OK】": CRT01
            End Select
    End Select
End Sub
'-----------------------------------------------------------
Public Sub Proc1() 'test pattern
    Dim rx As Double, rr As Double, p As Double
```

```
    Dperas
    Iproc = 1
    Icolor = QBColor(0) '(Black)
'   Icolor = QBColor(12) '(Red)
'   Icolor = QBColor(2) '(Green)
'   Icolor = QBColor(9) '(Blue)
'   Icolor = QBColor(14) '(Yellow)
    rx = 200
    For p = 1 To 0.2 Step -0.1
        Dpensz (Int(2 * p) + 1)
        rr = rx * p
        Dpmove rr, rr
        Dpdraw -rr, rr
        Dpdraw -rr, -rr
        Dpdraw rr, rr
    Next p
    Dptext 0, 0, (" Test Line Drawing")
    Dpcirc 0, 0, rr
End Sub
'-----------------------------------------------------------------------
```

5. Fortran からの VB への変換

5.1 Fortran ソースコードのファイル単位の修正

拡張子を変更して原稿にする

Fortran のプログラムコードを VB6 に書き直す場合、ソースコードをそのまま VB6 の標準モジュールとして読み込んでコンパイルに使います。エラーが出ますが、それを修正する作業手順をこの章で説明します。Fortran のソースコードはテキストファイルであって、その拡張子は通常(.for)になっていますので、それを(.bas)に名前を変更します。Fortran 言語では、サブルーチン名はすべてグローバル扱いであって、ファイル単位に分けるのは単なる便宜的なものです。ライブラリ単位でまとめるか、テキストが大きくなって単純なエディタで扱う量として大き過ぎる、などの時にファイルを分割していました。これに対して、VB6 のソースコード .bas ファイル（これを標準モジュールと言います）は、サブルーチンやファンクション（プロシージャと言います）を、相互に関連のある単位で集めます。或るサブルーチンが別のファイル（モジュール）に在るサブルーチンから呼ばれるときには、Public Sub **** と宣言しなければなりません。そのサブルーチンが同じファイル内から呼ばれるのであれば、Private Sub ****とします。Fortran のプログラムコードで、入出力処理を含まない部分は、単純な文字修正で済みます。ユーザインタフェースを含めて入出力が絡む部分は、フォームモジュールとの関係を考えなければなりません。

文字修正で済む変換

Fortran のソースコードを、単純に拡張子を(.BAS)に変更しただけのモジュールを組み込むと、リストは文法エラーの集合になります。そこで、テキストの編集機能を使ってまとめて書き換えができるものから修正作業を進めます。修正項目は、順番が重要であるものと順不同でよいものとがあります。

- Fortran の文字定数の引用符 ’ を ″ に変更します。
  先頭カラムが文字記号（通常は C）で始まる行はコメント行（注釈行）ですので、この文字を ’ に書き換えます。この修正には、上記の変更を先に行なっておきます。英字の C は文字として頻繁に表れますので、行の先頭であることを確認して文字を置き換えます。
- SUBROUTINE のキーワードは、Sub に置き換えます。
  Private, Public のキーワードのどちらをつけるかは後から考えることができます。分からなければ、最初は Private Sub にしておくのが安全です。実行のとき、外部から参照されれば、未定義コードのエラー通知が出ますので、そのときに Public Sub に変えます。
- SUBROUTINE 単位は END で終りますが、VB では End Sub または End Function に直します。
  この修正をすると、ソースコードの同一モジュール内で、複数のプロシージャの切れ目に横線が入って表示されます。
- RETURN は、Exit Sub または Exit Function に直します。
  ただし、プロシージャ単位の最後の行、End Sub などの直前にあれば省略します。
- DIMENSION は Dim に置き換えます。
- 論理演算の表記を、記号に置き換えます。これは下のように直します。
  .GT. (>)　　.LT. (<)　　.EQ. (=)　　.GE. (>=)　　.LE. (<=)
  .NE. (<>)　　.AND. (And)　　.OR. (Or)
- 組み込み関数などには、名前の変更だけで済むものがあります。
  例えば、SQRT, ATAN, DTAN などは、Sqr, Atn, Tan に書き換えます。ただし、これはプログラムの実行時に未定義関数または未定義配列などのエラーが出たときを待って変更することができます。

宣言文などの見直しと修正

Fortran では、宣言なしに使用する変数名は、最初の英字名を見て実数型か整数型かを決めることができます。IMPLICIT 文による定義は、サブルーチン単位ですべて書かなければなりませんでしたが、VB6 ではモジュールの先頭に、下の例で示すように Def***の形に書けば済みます。VB6 では、宣言無しの変数名はバリアント型になりますので、VB では変数名の型を明示的に宣言しなければなりません。このため、モジュールの先頭の宣言部(Declarations)に下で示す文を加えます。なお、配列の基底は VB6 では 0、Fortran では 1 ですので、配列を使う場合にはオプション文を使います。下の例は実数をすべて倍精度で扱う場合の例です。

```
Option Base 1
DefDbl A-H, O-Z
DefInt I-N
```

上記の修正方法は、あまり薦められません。できれば、モジュールの先頭で Option Explicit を宣言し、すべての変数の宣言文をまとめ、そこで変数ごとに型を指示します。Dim のキーワードに続けて、変数名個々の後に、As Integer, As String などを付けた表記に変えます。また、配列を使うとき、Option Base 1 を外すと、添え字 0 の変数領域が無駄になるだけです。丁寧に修正するときは、配列の使い方で、基底 1 からを 0 からに直す必要があります。

共通変数は Public のキーワードを付ける

Fortran で共通に利用する変数や定数は、COMMON 文を使うとサブルーチンの引き数を節約できます。この場合、COMMON 文を共有するサブルーチンをまとめて一つのファイル（標準モジュール）にするのが能率的です。この COMMON 文で利用する変数や配列名は、VB6 の(.BAS)モジュールの先頭の宣言部(Declarations)に載せれば、個々の Fortran サブルーチン単位のように毎回の宣言が必要でなくなります。変数名の付け方はプログラマの個性が反映されますので、他人が書いたプログラムコードを見るときの一つの障害になります。古い Fortran の文法では、変数名の長さが 6 文字以下の制限がありましたので、名前の綴りが全体として短く表記され、これがしばしば理解の妨げになります。モジュール単位、プロシージャ単位で使うローカルな変数名は、別のモジュール単位、プロシージャ単位で名前が同じであっても、原理的には別の変数名として扱われますが、これはなるべくなら避けるようにします。For...Next 文の繰り返し変数などに、I, J, K, などをローカル変数として使うのは普通の習慣です。名前に矛盾した使い方をしていないか否かを検査するには、モジュールの宣言部で、Option Explicit 文を入れ、変数名が Dim 文なしでは使わないようにして確認します。これは少し作業が鬱陶しくなりますので、プログラムコードに文法上のエラーが無くなって、実行時のテストの時に付け加えるのが良いでしょう。

データ初期値の設定

定数を定義する方法は、Fortran では PARAMETER 文があり、これは VB の Const 文に変更します。しかし、変数や配列に初期値を代入する Fortran の DATA 文は、代入文に書き直すか、Array 関数を利用します。そのため、初期値の設定などを行なわせるプロシージャ単位として Init()を作り、ユーザープログラムの最初に呼びようにすると分かり易くなります。

継続行の書き換え、マルチステートメント

Fortran の標準ソースコードは、1 行 72 文字までの制限があり、カラム（欄）の使い方が決まっています。先頭から 6 文字目の欄に何かの文字があれば、この行は前の行からの継続を意味します。VB の継続行の記号は、前の行の最後に、「スペース、アンダースコア」(_)を追加することで行なわせます。この変更は、面倒でもすべてのリストを眼で検査して修正しなければなりません。短いプログラムステートメントが複数行に亘って書かれているとき、VB6 ではコロン（:）を挟んでマルチステートメントにまとめることができます。ただし、行頭に、引き数の無いプロシージャ名を Call 宣言なしに書くと、処理名ではなくラベルと判断されますので注意します。

5.2 プログラムの制御--構造化プログラミング

GOTO ラベルを書き変えること

古い仕様の Fortran 文のコードやデータは、特定の行を識別するためにラベル（数字）を利用してプログラムコードを書きました。そのラベルを指定して、そこに制御を移すようにするとプログラムの流れが理解できますし、作業も便利でした。VB6 では、行頭ラベルに英字名が使え、コロン（:）を付けることでラベル名になります。しかし、GOTO 文を多用すると、プログラムのソースコードのリストを見て内容を理解しようとしても、制御があちこちに飛んで行きますので、コードを読むときに混乱してしまいます。そのため、行の上から下に自然にプログラムが記述できるような分かり易くプログラミングできる言語の開発へと進化してきました。これが構造化プログラミングです。この思想は、GOTO 文を無くしたプログラムコードを書かなければならない、とする過激な主張に表れますが、適度な GOTO 文の使用は望ましいことです。しかし、プロシージャ単位では、構造化プログラミングの思想は悪くはありませんが、イベント起動型のプログラムでコールバック処理があると、全体としてのプログラムの流れを追跡することが困難になってきました。制御の入口はプロシージャの先頭ですが、どこからそこに制御が入るのかの発信元が分からなくなってきました。

構文の変更が必要になる文

Fortranの計算型GOTO文やIF文と組み合わせたGOTO文などは書き換えが必要です。この書き換えは、論理構造の構成則の変更などと共に工夫をしなければなりませんので、かなり書き換えには手間を食います。個別の問題を以下にまとめます。

- 単純 If 文；例えば、If *condition* GOTO *label* : If *condition expression* のように一行で完結している条件文は、*condition* の後にキーワードの Then を挿入します。
- 上記で、expression 部を次の行に写す場合には、Then の後で改行して、expression 部の後でさらに改行し、End If で条件文を閉めます。この方式では expression 部に複数行に亘るコードが記述できます。
- If 文と関連して複数の GOTO 分岐がある場合には、If Then, ElseIf, Else, End If を使って、できるだけ GOTO 文とラベルとを省きます。
- GOTO (label_1, label_2, …), *Number* の構文は、Select Case 文を使って書き直します。
- DO label … CONTINUE の構文は For...Next 文に書き直します。DO 文が入れ子になっているとき、また、ループの中身が多いと、For と Next との対応が分かり難くなりますので、ラベルなどはコメントとして併記し、Next になるべく変数名を付けて利用するようにしておくとリストが見易くなります。また有意のブロック単位で全体のプログラム行をインデントするのも役に立ちます。ただし、VB ツールのテキストエディタでは、インデントは自動的には設定されません。
- VB の制御文には、For...Next の他に While...Wend、Do...Loop などの書き方が利用できますので、内容によって選択します。ただしプログラムの中身が理解できないと正しいコードが書けません。

Format$ と Str$ の使い分け

入出力の処理で現れてくる Fortran の FORMAT 文では、文字並びが等幅フォントを前提としていましたので、書式の制御は比較的単純でした。文字の位置や印刷の幅は、文字位置のカラム数を踏まえて、計算した通りに表示することができました。これによって、例えば、数表を印刷させるとき、小数点の位置を縦に揃えることができました。しかしながら、Windows ではリストに使う文字は字体によって字幅の異なるプロポーショナルフォントを標準としますので、印刷時の書式制御文が意味をなさなくなりました。ただし、等幅フォントも使えないと不便です。明示的に等幅フォントを使いたいときには FixedSys フォントを指定します。ただし、このシリーズはフォントサイズの種類が少ないのが欠点です。数値をリストするとき、小数点以下何桁まで表示させる、などの出力制御は必要です。この指定をするため、VB では数値を文字型に変換する関数に Format$関数が加わりました。もし、なにも書式制御を必要としなければ、Str＄関数を使うのが便利です。

6. N88Basicのコード書き換え

6.1 ファイルの書換え

Quick Basic よりも古い言語構造であること

N88Basic の言語構造は、もともと 8 ビットのプロセッサを対象として出発し、上位互換を保ちながら機能を充実してきましたので、プログラミング言語としては欠陥が多くあります。VB6 への書き換えは、この欠陥を修正することも主要な作業です。これは種々の問題があります。まず、モジュールファイルの考え方が異なります。N88Basic は、全体を一つのモジュールファイルに集約します。この中に、サブルーチンのブロックを組み込んで、GOSUB-RETURN で利用しました。幾つかのファイルに分散する場合には、実行前は MERGE 文で、実行中は CHAIN 文でファイル単位のソースコードを取り込みます。Quick Basic にあるような、サブルーチンをプロシージャとして独立させ、複数のモジュールファイルに分ける、という考え方が未だ育っていなかった頃の習慣を引きずってきました。したがって、考え方としては、N88BASIC のソースコードを Quick Basic のソースコードに変換する作業と共通するところがあります。

変換作業の手順

N88Basic のソースコードは、そのまま VB6 の作業画面に呼びだしてコンパイルします。このとき、エラーを表示する赤色のコードで埋まりますが、エラーでないコードもかなりの比率で残ります。したがって、この赤いコード部分を書き直せばよいことになります。これには、VB6 の編集メニューで、検索または置き換えの機能を利用すると作業がはかどります。まず言語仕様の基本的な事項について説明します。

- N88Basic は、全体が一つのプロシージャ単位になっていて、ファイル名がプログラム名を兼ねています。プログラムの入口は先頭行でしたので、特に明示する必要がありませんでした。VB6 では入口のプロシージャ名が必要です。標準として付けるならば、Public Sub main() を頭につけ、リストの最後の行に End Sub を追加します。なお、実際のコードでは、Public Sub main() の行に先行して宣言部が並びます。
- 一つのプロシージャ内で使う GOSUB-RETURN 文は VB6 でも有効ですが、なるべく明示的にプロシージャ単位に分割するように、Private Sub/End sub で区切って、ローカルなプロシージャ単位に直します。作業としては、ソースコードで、"GOSUB " を"Call " に置き換えます。そして、*で始まるラベル名に Private Sub を付け、"RETURN" を End sub に置き換えます。GOSUB に数字ラベルを使っているときは、飛び込み先に英字のラベル名をつけてプロシージャに仕立てます。実際のコーディングでは複数の RETURN 文が使われることもありますので、プロシージャの終わり以外は Exit Sub に直します。
- 行番号付きのソースコードは、そのまま使えます。ただし、IF 文などでは、明示的に GoTo を追加しなければなりません。
- 幾つかのプロシージャの集合は、処理のまとまり毎に別のモジュール（.bas をつけた別ファイル）に分けます。この際、別のモジュールから呼ばれるプロシージャ名は Public Sub とします。共用する変数名は、どちらか一方のモジュールで、コモン変数名であることを明示的に Public 宣言をしなければなりません。

6.2　変数名についての注意事項

N88Basic は、小さなプログラムをユーザーが簡単に作れるようにした言語ですので、高級言語の感覚から見ると、かなり曖昧な仕様になっています。これはユーザーの使い勝手を上げますが、これが VB6 への書き換えで引っ掛かり、エラーの原因になります。その大部分は、変数名の扱いにあります。

- 変数名は明示的に型の宣言を必要とします。VB では、型の宣言を省いた変数名は Variant 型として扱われますので、実用的にはあまり混乱が起きません。しかし、科学技術計算では変数の型を正確に区別しなければなりません。これには Fortran の習慣を引き継ぎます。モジュールの先頭の宣言部には、例えば下のように書き加えます。変数の使い方が上の宣言と異なる場合には、その都度、型の宣言文を追加します。

  ```
  Defint I-N
  DefDbl A-H, O-Z
  ```

- 8 ビットのプロセッサを使っていたマイクロコンピュータ用の Basic 言語では、英字で２字までの変数名しか使えませんでした。そのため、例えば、型を区別する変数名として、A%, A!, A#, A$ を別々の変数名とすることができました。また、単独の変数名と配列の変数名、例えば AA と AA()とは別の変数として扱うことができました。これは VB では許されませんので、変数名の付け替えが必要です。
- 変数名には特殊記号(%, &, !, #, @, $) を使いませんが、これまでの習慣として、型識別の接尾文字を付けられます。例えば、文字型を表す C$のような使い方もできます。しかし、明示的にデータ型の宣言（例えば、Dim C As String）をする方が確実です。
- 変数名の使い方として、文字数によって変数のスコープ（通用範囲）を決めておくのは実践的な知恵です。例えば、I, J, K などの１文字変数名はローカル変数とします。これらは、For...Next 文などで局所的に使われる変数名の定番になっています。
- 技術計算の習慣で、１文字の変数名を使いたくなります。例えば、b, d, E, h, M, n, N, P, Q, S...などの記号は、幅、距離、ヤング率、高さ、モーメント、ヤング率比、軸力、荷重、剪断力... の記号としての定番です。これらは、より分かりやすい長い名前に直して使います。
- 変数名には、VB の言語仕様で使われている英字名が使えませんので、言語レファレンスの索引に表れる名前は避けます。これをキーワードと言うのですが、この他に予約語があります。うっかり使う頻度の高い名前に次のような２字の単語があります。

  AS, AT, DO, IN, IS, ON, OR, TO, VB

- 一つの実践的な方法として、変数名は最少で 3 文字に直します。4 文字使うと、かなり区別し易くなります。VB では変数名の大文字・小文字を区別しませんが、読み易さを考えて大文字・小文字を使い分けることができます。
- 変数名は、VB では 255 文字の長さまで許されていますが、あまり長いと読み難くなります。二つ以上の単語の合成で名前を付けるときには、間をアンダースコア（_）で繋ぐ習慣が多くなりました。なお、長い行を分割するときは、行末に（スペース、アンダースコア）を継続行の記号として、残りを次の行に続けることができます。
- N88Basic では、一つのファイル単位が一つのプロシージャの扱いになっていますので、大きなプログラム単位であっても、変数はすべて共通変数の扱いです。そのため、同じ変数名を別の場所で別の目的に使う場合でも、データの干渉が無いように使うこともできました。しかし、そうすると、サブルーチンをプロシージャ単位に独立させ、さらに別のモジュールに分けるとき、変数定義の場所と参照の場所を意識しておかなければなりません。一つのモジュール内の共通変数は、モジュールの先頭の宣言部に Dim 文で宣言して集めます。もし別のモジュールでも参照される場合には、Public を付けて宣言します。プロシージャ内でローカルに使う変数は、そのプロシージャ内で宣言して使います。このテストは、実行の段階で改めて見直しします。
- 変数に初期値を代入するには、N88Basic では READ/DATA 文の組みを使いました。VB では、このキーワードが使えませんので、少し変わった方法で対処します。まず、Variant 変数名を定義しておいて、Array 関数で初期値を代入します。この Variant 変数のデータは配列として作成されますが、変数名の定義では配列の寸法を指定する必要がなくて、Array 関数のデータ個数分の寸法が自動的に設定されます。所定の変数に代入するには、Variant 変数のデータ値を指定の変数の型に合わせるようにして参照しますので、二段構えの処理になります。なお、Array 関数で代入される Variant 型の配列では、Option Base 1 の宣言が効かないとマニュアルに書いてありますが、実際に試してみると、この宣言が適用されていることを発見しました。

6.3 幾つかの制御命令の書き換え

プログラムの実行時の制御命令に使われるキーワードの中、修正や書き換えが必要な項目を解説します。

- IF...THEN...ELSE 文は、複数行に展開し、End If 文で閉じるように変更します。一行で納まる短い If 文はそのままでも有効ですが、End If 文で閉じるような構造化が推奨されています。If 文の中でジャンプ先を指定するとき、ラベルだけの記述でなく、GoTo ラベルと補わなければなりません。この書き換えのとき、成るべくなら GoTo 文を使わないように論理を組み立て直すのが望まれます。そうすると、番号ラベルが必要でなくなります。論理式を書き直すときは、思い違いをし易いので、最初の段階ではラベルを生かした構文にしておいて、動作を確かめながらラベルを外すようにします。If...Then...Else...End If 文は、構造化が分るようにソースコードにインデントを使って見易くまとめます。これは VB のコード編集機能にはありませんので、ユーザーが工夫しなければなりません。
- ON...GOSUB/ON...GOTO 文は、Select Case...End Select の構文に直します。後者の方が使い勝手が良くなります。
- プログラムの実行制御で、GOSUB 文は、サブルーチンを呼び出す Call 文に直しますが、呼ばれる側のサブルーチンプロシージャは、さし当たっては引き数を持ちません。そのため、ここで利用する変数と結果を持ち帰る変数とが不定になっています。これをテストするには、モジュールの始めの宣言部に Option Explicit を付けます。そうすると、すべての変数は明示的に Dim 文での定義を要求しますので、これを見て、サブルーチンで引き数として利用する変数名を共通変数としてモジュールの始めの宣言部に定義していきます。サブルーチン内部で作業用に使う変数は、このサブルーチン内で定義します。この作業は、変数名を検索して、どこで使われているかを確かめながら進める必要があります。引き数としての変数の利用が限定的であるものは、仮引き数を定義したサブルーチンにする方が勝ります。
- N88Basic では、一つのモジュールが一つのプログラム単位になっていて、別の処理を実行させたいときには CHAIN 文を使います。この機能と同等のことを VB で行わせるには、Shell 関数を使います。これは、ある実行時のプログラムから、別のプログラムを起動するコードですので、元のプログラムもそのまま実行が継続されます。したがって、元のプログラムを意図的に終了させないと、メモリ上には二つのプログラムが同居します。これが CHAIN 文と異なる点です。16 ビットのマシンの時代には、元のプログラムを部分的に書き換える Overlay の技法が内部的に応用されていましたが、メモリの制限のない 32 ビットプロセッサでは必要がなくなりました。
- VB6 は、未完成のコードであっても、エラーが無い場所まで実行させることができます。この機能が他の言語よりもプログラム開発を便利にしています。実行には幾つかの選択肢があります。最初は単純に実行させてエラーの出具合を観察します。「完全コンパイル後に開始」で実行させると、当面必要でないコード部分を含めてすべて構文エラーを検査してくれます。
- プログラムのデバッグは、プログラムの中身についての理解を含め、さまざまな知識の総合が必要ですので、定型的に説明することはできません。しかし、Basic 言語が便利であることの最大の理由は、いわゆるダイレクトモードで実行ができることにあります。VB ではイミディエイトウインドウを使って、そこにコードを直接書いて実行できる機能として提供されています。例えば、プログラムを実行してエラーが起きたとき、問題になっている変数の値を Print 文で表示させることができます。
- プログラムのテスト実行をさせると、場合によって無限ループにはまったり、意図しないエラーで作業が中断されたりする危険があります。そのため、こまめにプロジェクトの保存をしながら作業をすることを勧めます。