"Plain_Basic"を使う
コンピュータプログラミング入門

はじめに

Plain_Basic は、非常に単純化した Basic 言語のインタプリタに著者が付けた名称です。コンパイラ形式の本格的なプログラミングツールではなくて、プログラミングの入門教育に使う教材として作成しました。Plain_Basic のモデルは、1980 年代に爆発的に普及した８ビットのマイクロコンピュータ（８ビットマイコン：microcomputer。my computer ではありません）に組み込まれていた Basic インタプリタです。この Basic 言語の仕様は、今は無くなりましたが「JIS 基本 Basic(JIS C6207-1982)」にほぼ準拠しています。当時、多くの人は、８ビットマイコンを使ってプログラミングを覚え、これがその後のコンピュータ関係の教育に大きな励みになりました。現在のパーソナルコンピュータ（パソコン：personal computer、言わば独り占めで利用するコンピュータ）の環境では、このような単純なプログラミングツールを見なくなりましたので、プログラミング入門の教材として作成しました。

大抵の人は、電卓（卓上電子計算器）の使い方を知っていると思います。実は、電卓はれっきとしたコンピュータです。パソコンを使っていても、電卓を手元に置く人は少なくありません。余計なものが殆どついていませんので、関数電卓を使いこなして、一寸複雑な数式の数値計算を進めるのは手間が掛かります。その大部分は、データの入力と結果を書き写す道具が不便であるためです。Plain_Basic は、電卓にプログラミングと記録の出力機能を持たせたと考えるとよいでしょう。出力には、グラフィックスを使えるようにしました。

８ビットマイコンの Basic 言語仕様は、その後に続いた 16 ビットパソコンに引継がれ、さらに Visual Basic などに発展して、初期の単純さが全く無くなってしまいました。言語構造としては、大型コンピュータで利用するコンパイラ形式の Fortran に較べて、能力が足りない不満がありましたが、ユーザインタフェースの機能には学ぶべきことが多くありました。そこで、著者は、1980 年代の後半に大型コンピュータの環境で、マイコン Basic 並のインタフェースを持つインタプリタツールを、Fortran 言語を使って開発し、これを NUCE_BASIC と名付けました。このツールは、Fortran で書かれた多くのサブルーチンを、ユーザがマイコン Basic 言語の特殊コマンドのような感覚で利用することが目的でした。特に、幾何モデリングのツール GEOMAP を組み込んだバージョンは、教育利用として好評でした。

Plain_Basic は、Fortran 版の NUCE_BASIC を、C/C++言語を使って書き直したものです。開発ツールには Borland C++Builder を使いました。パソコンの OS 環境が 32 ビットの Windows 系が主流になりましたので、この環境で、昔の８ビットマイコンのインタフェース環境を再現するように開発しました。Plain_Basic は、初心者のプログラミング教育に利用しますが、C/C++で作製した本体のソースコードは、やや高度なプログラミング教育に利用することを考え、そのソースコードも教育に利用することにしました。また、この Plain_Basic をプロトタイプとして幾つかのインタプリタを開発しました。主要なものは、幾何計算用のツール Geometry_Basic、幾何モデリングツールを組み込んだ GEOMAP_Basic があります。これらのマニュアルは別にまとめました。

2006 年 8 月 12 日初版：島田　静雄（全 72 ページ）

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。

---

1. 基礎知識の穴埋め

1.1 算術の常識を再確認する

引き算は難しいこと

殆どの人は、小学校の低学年で教わる、1+1=2 に始まる整数の加減乗除の算術を、どうやって覚えたかの過程を忘れていると思います。一桁の足し算は一年生で習います。掛け算は、足し算の知識の上に成り立ちます。同じ数を何度も繰り返して足す結果を「九九」で覚えます。引き算は、かなり遅れて二年生になって習うのですが、その理由は足し算の知識が応用されるからです。二桁以上を扱う引き算は、引けない位のとき、上位の桁から 10 を借りてくる方法を習います。しかし、全体として引けない引き算の問題を含ませてありません。**負の数**の概念は、小学生には難しいからです。算術の割り算は、引き算を繰り返し応用することですので、引けなくなった残りが**余り**です。電卓が使える現代では、「九九を始めとする算術を覚える必要がない」とする過激な論もあります。しかし、市販の電卓で「10÷3 の答は 3、余りが 1」の計算をしようとなると、一寸した計算手順を工夫しなければなりません。普通の電卓で余りの計算、さらには**過不足**の計算が一回で出来ないと言うことは、一般の人には、案外、常識の盲点になっています。

算術は整数の計算を扱う

電卓で「10÷3」の計算をさせると、3.333…と得られます。これを**実数**表現と言い、答を正確に表そうとすると無限に数字を書かなければなりません。小学校では、実数の代わりに**分数**を習います。数学の計算では整数と実数とを区別しませんが、実社会では無限に数の並ぶ実数を扱うことをしません。小数点を含む数であっても、小数点の位置を便宜的なものと考えて、小数以下を何桁かで区切り、そこに最小桁のある整数扱いをします。したがって、乗除計算の約束は、整数と実数とで扱いが異なります。社会生活で使う数は、実質的にはすべて整数として扱いますので、電卓で一意に計算をさせるとき、実質的な単位がどこにあるかを判断しなければなりません。これが**切り上げ**や**切り捨て**の処理の背景です。お金の計算では余りを切り捨てることが常識ですが、科学技術関係では**四捨五入**が使われます。日本工業規格では、少し工夫を凝らす **JIS の丸め**を提案しています。これは、末尾が５で終わる数の四捨五入は、その前の数が偶数になるように切り捨てるか切り上げるかを決めます。科学技術に関係する社会では、レポートなどに書かれている数字の表現方法を見れば、そこの技術レベルの成熟度が判断できます。例えば、円周率を表すとき、3, 3.14, 3.1416, 3.141592 などを、状況に応じて使い分けます。桁数を多く並べれば正確である、と思うのは素人考えです。

算盤は正の整数だけを扱う道具であること

算盤（そろばん）の歴史は古く、基本的な使い方を覚えれば、便利に利用できる日常計算の道具です。電卓があっても、算盤に手の行く人は案外多いのです。算盤は、原理的には正の整数だけを表示します。引き算をしていって、引けなくなったときは、同じ算盤で仮に別の桁位置を使うか、別の算盤を使って、引く側と引かれる側の数を入れ替えて計算します。ところが、電卓ではマイナスで表示します。このとき、**負の数**の概念と、**過不足**の概念が混じりますので、それを合理的に解釈しないと混乱します。数学の環境では二つ以上の解釈が出る方法を嫌いますので、負の数の表現を優先します。しかし、実生活では、負の数の概念と余りとは混在して使われています。お釣の計算は余りですが、利益の計算では、負の数を赤字と呼んで区別します。布や紙を一定寸法で区切るときの余白部分が余りですし、寸法不足分を言うときは概念的には負の数です。

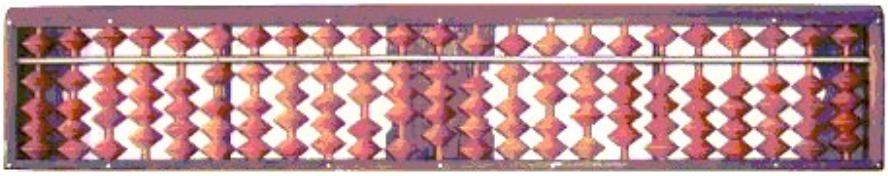

図 1.1　算盤は一続きのレジスタを適当に区切って正の整数の表示をする道具です

補数の表示方法を算盤で理解する

算盤で足し算・引き算を連続して計算するとき、引けなくなった場合でも計算を連続的に進めたいことがあります。このとき、一時的に頭の方に 9999…を詰めた表示を使うことがあります。これが**補数(complement)**です。ただし、補数を生の数字に書いて扱うことはしません。算盤の場合は 10 進数の補数ですが、コンピュータのレジスタでは、負の数を表す方法として２進数の補数を**内部表現**で使います。算盤の常識があると、補数の表現を納得できます。しかし、そうでないと現実感がなくて、途端に難しい問題になります。算盤で考えると、負の数の補数表現は９を頭の部分に埋めるのですが、適当な桁区間を考えて、そこの頭に９を詰めます。この、適当な区間に区切る単位が計算機械では決めてあって、これが**レジスタ**です。単純な電卓では整数 10 桁が普通です。コンピュータでは２進数の桁（ビット）で言い、内部では 8/16/32/64 ビット単位のレジスタで扱います。算盤は一続きのレジスタを、使う人が適当に区画を決めて、複数のレジスタに分けて使います。コンピュータの場合には、逆に、単位長さのレジスタ（例えば 16 ビット）を論理的に繋いで桁数の多いレジスタとして使います。

1.2　計算機から電卓まで

電卓の前に機械式の計算機があったこと

パソコンが便利に利用できる環境であっても、一寸した数値計算には電卓が便利です。そのため、Windows のパソコンには、アクセサリとして電卓が使えるようにもなっています。コンピュータが使えなかった時代、桁数の多い数値の乗除計算を正確に処理することは、大変な苦労でした。特に、お金にからむ計算は、正確な整数計算を基本としているからです。算盤で、桁数の多い数の掛け算・割り算をするには、相当の技能と熟練が必要です。これを助けるために**機械式の計算機械**を使いました。これは、当時の物価水準から見れば、非常に高価でした。計算機械は、銀行など、お金の計算をするところに主に需要がありました。設計計算など、科学技術にからむ数値計算が必要である企業・大学・研究所でも必須の備品でした。電動の計算機は、手回しの計算機を高速化した装置ですが、当時の物価水準では高級乗用車くらいの価格でした。

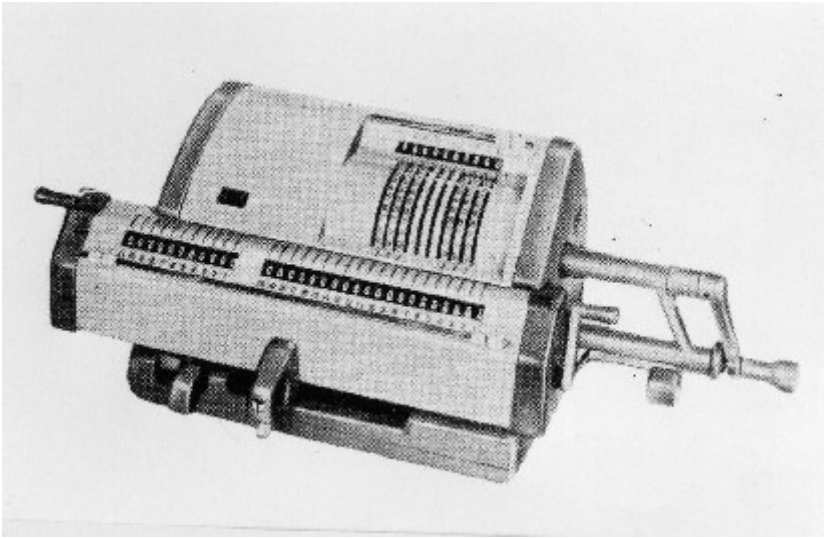

図 1.2　10 桁×10 桁の計算ができる手回し計算機（タイガー）

計算尺はアナログ式乗除計算の道具である

科学技術関係の数値計算をするとき、計算機械が使えなかった環境では、桁数の多い数値の乗除計算は大変でした。その一つの方法は、**常用対数表**を使って実数を対数に直し、その加減算を算盤で行い、再び対数表を逆に使って結果を数に直します。しかし、一寸した技術計算で、精度の扱いに神経質でなければ、計算尺を乗除計算に使いました。sine, cosine などの基本的な数学関数の値を参照するには**数表**が必須でした。地形測量の計算では、sine, cosine の値を、更に常用対数化した分厚い数表を使いました。コンピュータが利用できなかった時代、技術者が使う数値計算の三種の神器は、「算盤・計算尺・数表」でした。今では、これが関数電卓に置き換わり、さらにパソコンの利用へと変化してきたのです。常用対数表を始め、種々の基本関数の数表が不用になりましたし、高校の数学教育でも対数を使う実用的な数値計算の問題を殆ど見なくなりました。

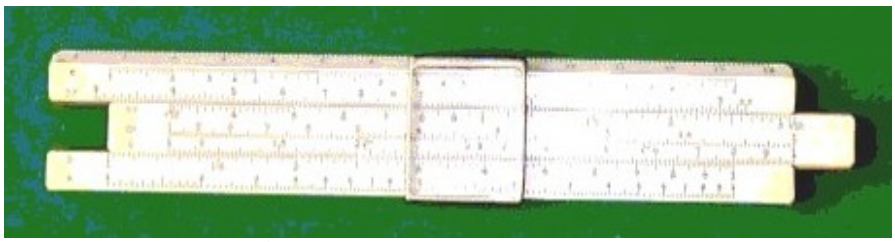

図 1.3　計算尺は技術者の必需品でした（ヘンミ）

電卓の内部構成は機械式計算機の道具立てと同じである

手回しで操作する機械式の計算機械（図 1.2）は、今では殆ど見ることができませんが、基本的な算術が出来る道具立てで構成されています。例えば掛け算は、数式で書くと「Ａ＝Ｂ×Ｃ」ですので、数値が入るレジスタをＡ、Ｂ、Ｃに対応して３つ使います。10 桁×10 桁の掛け算の計算結果は最大 20 桁になりますので、Ａの表示桁数はＢとＣの表示桁数の和で構成します。電卓の場合には数の表示窓が一つしかありませんが、内部に三つのレジスタがあって、計算手順に合わせて表示の内容を切り替えています。非常に単純な 10 桁の電卓では、三つの内部レジスタはどれも 10 桁で扱いますので、例えば、整数だけの６桁×６桁の計算をすると結果が 10 桁を越えてエラーになります。ただし、少し高級な電卓では、**指数表示**に切り替えて表示して、エラーになりませんので、安価な電卓を使わないと、この確認ができません。小数を含む数の掛け算では、計算桁数の頭 10 桁が表示され、それ以降は捨てられます。ただし、整数部分の桁数が 10 桁を超えるとエラーになります。整数を使う割り算では、計算方法が実数に切り替わり、もし割りきれないと 10 桁全部を使った数が表示されます。例えば、10÷3 = 3.333…となるのがそうです。このように、電卓では、整数計算と実数計算とが混ぜこぜに応用されていますので、算盤や機械式計算機を使う場合とは少し違った計算術を応用します。

図 1.4　Windows のアクセサリには電卓が含まれています

電卓では補数の表示方法がないこと

機械式の計算機（図 1.2）で、０引く１の計算をすると、レジスタにズラッと９が並びます。電卓で、同じ計算をすると、マイナス記号の表示と共に単純に１が表示され、補数表現は出ません。電卓のレジスタは、表示する桁数に加えて、符号を保存する桁が内部にあります。一方、コンピュータは**二進数のレジスタ**を使いますので、整数を表現するときの約束が二通りあります。一つは正の整数だけを表す場合であって、16 ビットのレジスタは 0 から 65535 までの数を扱うとします。この最大数に１を足すと 0 に戻ります。もう一つの約束は、先頭の１ビットを負の符号に約束します。16 ビットのレジスタは 15 ビットまでを有効な数値桁として扱いますので、**正の整数の最大値**は 32767 です。負数は二進数の補数で表し、０から１を引くと、すべてのビットが１になります。**負数の絶対値の最大**は、先頭のビットだけが１であって、この場合には-32768 です。この状態から更に１を引くと、物理的に先頭のビットが０、残りがすべて１になりますので、これは正の整数の最大値を表します。この約束は、ビット並びを数字で表すときの約束ですが、内部的な計算規則はどちらも同じです。このビット並びを符号付きの整数とするか正の整数とするかの約束がコンピュータ言語の**型の定義**です。

電卓はメモリを一つ使う

電卓にはMの字のついた処理キーが幾つかあります。これは、一つの**加算レジスタ**が別にあって、これを**メモリ**として使います。主な利用は**総計計算**(grand total)の記録です。コンピュータの場合には、メモリを内部的に何メガの単位で準備しますが、利用方法の原点は電卓のメモリの使い方と同じです。したがって、コンピュータのプログラミングを考えるとき、電卓の１個のメモリを使いこなす手順を理解しておくことが役に立ちます。電卓はれっきとしたコンピュータですが、その基本構成は三つのレジスタと一つのメモリです。パソコンのプロセッサは、この延長にあって、レジスタの数も種類も多く、メモリを多く使うことができ、レジスタを扱う多様な命令が準備されています。

1.3　計算は技術であること

算術と数学は実践と理論の関係にある

算術と言う言葉を使うと小学校で習う初歩的なレベルを想像しますので、体裁を付けて言うときは「**数値計算法**」です。しかし、これは学問としての「**数学**」とは違って「**技術**」に属します。そのため、理科系の大学では、工学部の方で主に研究されています。理屈っぽく説明しますが、技術は三つの要素で構成されます。「**道具**・**技法**・**技能**」です。計算の道具は種々の工夫が凝らされて進化しています。算盤・機械式計算機・電卓・コンピュータの歴史がそうです。技法は道具の使い方や原理などを指し、ここに数学も位置付けます。コンピュータの用語で言うと、道具と技法はハードウェアとソフトウェアに当たります。技能は個人個人が自分で覚えて身に付ける能力を指します。コンピュータ利用の見方から言うと、キーボードの操作に慣れることは技能の一つです。**技術移転**(technology transfer)では、道具と技法は眼に見える物として伝えることができます。しかし、道具を使いこなす技能は人と人との繋がりで伝える部分が主ですので、その繋がりが切れると技術全体が死滅することも起こります。現代は書店に行けば多くの参考書がありますし、インターネットを介すれば個人の環境で多くの知識つまり技法が得られます。しかし、学校での対面授業のように、人との交流の場を大切にしないと、技能の実際を納得することはできません。

タイピングは基本技能である

一般教養は、「**読み・書き・算盤**」と言われます。コンピュータはアメリカ主導型で開発・進歩してきましたので、「読み」は英語の素養を言います。説明には英語を語源とするカタカタ語が氾濫しますが、分からないときは**コンピュータ用語辞典**で当たって下さい。「書き」方は範囲が広く、プログラミングはプログラミング言語を使う一種の**英作文**です。マニュアルなどは日本語で書く文書です。どの場合でも、**テキストエディタ**や**ワープロ**を使うことになりますが、タイピングが基本技能として必須です。上手になるには自分で技能を磨くしかありません。参考書に書いてあるのは打ち方の技法です。読んだら忽ち上手になるものではありません。道具として使う「算盤」に当たるコンピュータは、それを理解するには少し奥行きが深過ぎますので、次節に電卓の使い方から解説します。

1.4　電卓の使い方を理解しておく

筆記用具が必要であること

あまりにも当然と思うため、算盤や電卓を使うときに、紙と鉛筆に代表される**筆記用具**のことを注意しないことがあります。現在のコンピュータの利用環境では、筆記用具を使わなくても済むことが多くなりました。逆説的に言うと、手で書くことを省くようにコンピュータが進歩したのです。手書きで綺麗な文書を作成することの技能に代わって、キーボードの使い方の技能が必須になりました。しかし、昔風に、手書きでも文書を作成する技能を弁えておく素養が大切です。その実際は、電卓を使って計算を進めるときに起こります。

数の入出力のときに間違いを起こし易い

算盤や電卓を使って実践的な数値計算をするとき、一番時間の掛かる処理は、数字をセットし、また計算結果を紙などに書き写すときです。この処理部分を**入出力**と言います。きれいで読み間違えをしないように数字を書くことは技能に属し、その訓練は一種のお習字に当たります。また、入出力の作業のときに最も間違いを起こし易いので、大事な計算では読み合わせや検算の処理が欠かせません。その時間は元の作業と同じくらい掛かります。無駄なように見えますが、安全のため、同じ計算を複数の人に独立にしてもらう方法も使います。この部分を合理的に計画する方法が種々工夫されています。その一つが印刷記録の残せる計算機の利用です。商店で普通に使う**キャッシュレジスタ**には簡単な**記録計算機**が使われています。科学技術計算に応用できて、印刷記録が残せる計算機械は無くも無かったのですが、非常に特殊でしたので、広く使われることはありませんでした。

計算書は検算が出来るようにまとめる

計算結果の自動記録ができない算盤や電卓を**一過性の計算機**と言います。幾つかの手順を必要とする計算では、すべての計算結果を書き出すのではなく、重要な節目ごとに結果を書き残しておいて、計算間違いを検査できるようにします。検算は、必ずしも元の方法と同じである必要はありません。これも技術ですので、検査を業務とする職種もあります。コンピュータは印刷機能が使えることが電卓よりも便利ですので、コンピュータを利用し始めの頃は、必要以上に大量のリストを作っていました。この熱が冷めると、反動で殆どリストを出さない利用をするようになり、結果としてコンピュータ処理が**ブラックボックス**化するようになりました。電卓の利用における煩雑さと、コンピュータ利用の便利さとを折衷する方法が数値計算プログラミングの技術です。プログラミングは、いわば、電卓をどのように利用するかの手順を文書化したものになります。したがって、プログラム文を見れば、計算の手順を追いかけることができますので検査に使うことができます。しかし、そのプログラムを全く見せないように隠してしまうとブラックボックスになります。

Plain_Basic はプログラミングのできる電卓として使う

全くの初心者を対象としてコンピュータプログラミングの手ほどきをしようとなると、受講者の知識レベルを考えて**カリキュラム**（教科のプログラム）を工夫しなければなりません。筆者は、何年もの試行錯誤をしました。自分がどうやってプログラミングを覚えたかの経過を振り返って見ました。その見方に立つと、初心者の置かれた位置は、コンピュータが使えなかった頃と状況が重なります。そして、筆算・算盤・手回し計算機・電卓の順を経て、８ビットのマイコンに搭載されていて Basic 言語で**プログラミング**を覚えたのでした。そして、８ビットのマイコンは、プログラミングのできる電卓の性格を持っていました。現在のパソコンは非常に高機能になりましたので、Basic を標榜する**プログラミング言語**は、今や高級言語に変質しましたので、初心者が利用するにはハードルが非常に高くなってしまいました。つまり、初心者にプログラミングの手ほどきに使うツールとしては、適当ではありません。そこで、このギャップを埋めるため、パソコンの環境で、マイコン時代の Basic の利用を再現するツールを考えました。これが Plain_Basic です。この使い方の手始めは、**関数電卓**を少し高級化したように工夫しましたので、初心者の利用入門にはハードルが低くなるでしょう。また、プログラムの経験が豊かな人にとっても、かなり実用的に利用できると思います。

2.　簡単なテュトリアル(tutorial)

2.1　パソコンの環境

実行形式のプログラムは Pbasic.exe

Plain_Basic は、Windows のパソコン環境で、擬似的な DOS の環境を作成して動作するように、Borland の C++Builder 5 を開発ツールに使ってコーディングしました。プログラム本体は 650 KB の寸法です。マニュアルは、コンパイル済みの htmlhelp 形式(*.chm)で準備し、HELP メニューで参照します。ただし、このヘルプファイルは、間接的に Internet Explorer を呼んでいます。通常の Windows 系のパソコンでは問題は起こりませんが、他のインターネットツールの環境では読めないことがあります。例題プログラムなどを含め、全体はフロッピーディスクで配布することができます。

二つのモニタ画面

Plain_Basic は、MDI 形式で三つの標準的な Windows 画面を使います（図 2.1)。全体枠が親ウインドウであって、上から**タイトルバー**、**メニューバー**、一行分の**テキスト入力枠**があり、その下に二つの子ウインドウ；コンソール出力用**テキストウインドウ**、**グラフィックスウインドウ**：が並びます。テキスト入力枠とテキストウインドウは、DOS のパソコン環境(CUI)を再現するように設計されていて、グラフィックスウインドウを別に追加した構成です。それぞれのウインドウの位置と寸法は、マウスを使って自由に設定できます。テキスト入力枠をアクティブにすると、キーボードから文字入力を受け付けます。改行でその文字列が読み込まれ、テキストウインドウにエコー表示されます。その文字列は、解読されて何かの処理が行われ、結果がテキストウインドウに表示されます。マウスによる操作やメニューの選択などの GUI 操作は、最小限の種類に抑えてあって、オプション(追加)的な処理に使います。Plain_Basic の終了は、コマンドの **QUIT** で行います。親ウインドウ右上の閉じるボタン×をクリックしても、また左上のシステムアイコンのプルダウンメニューで閉じる指示も使うことができます。プログラムが暴走したとき、脱出する方法（例えば Esc キー）が利かないときは、Windows システムの再起動(Ctrl+Alt+Del キー)を使うことになります。

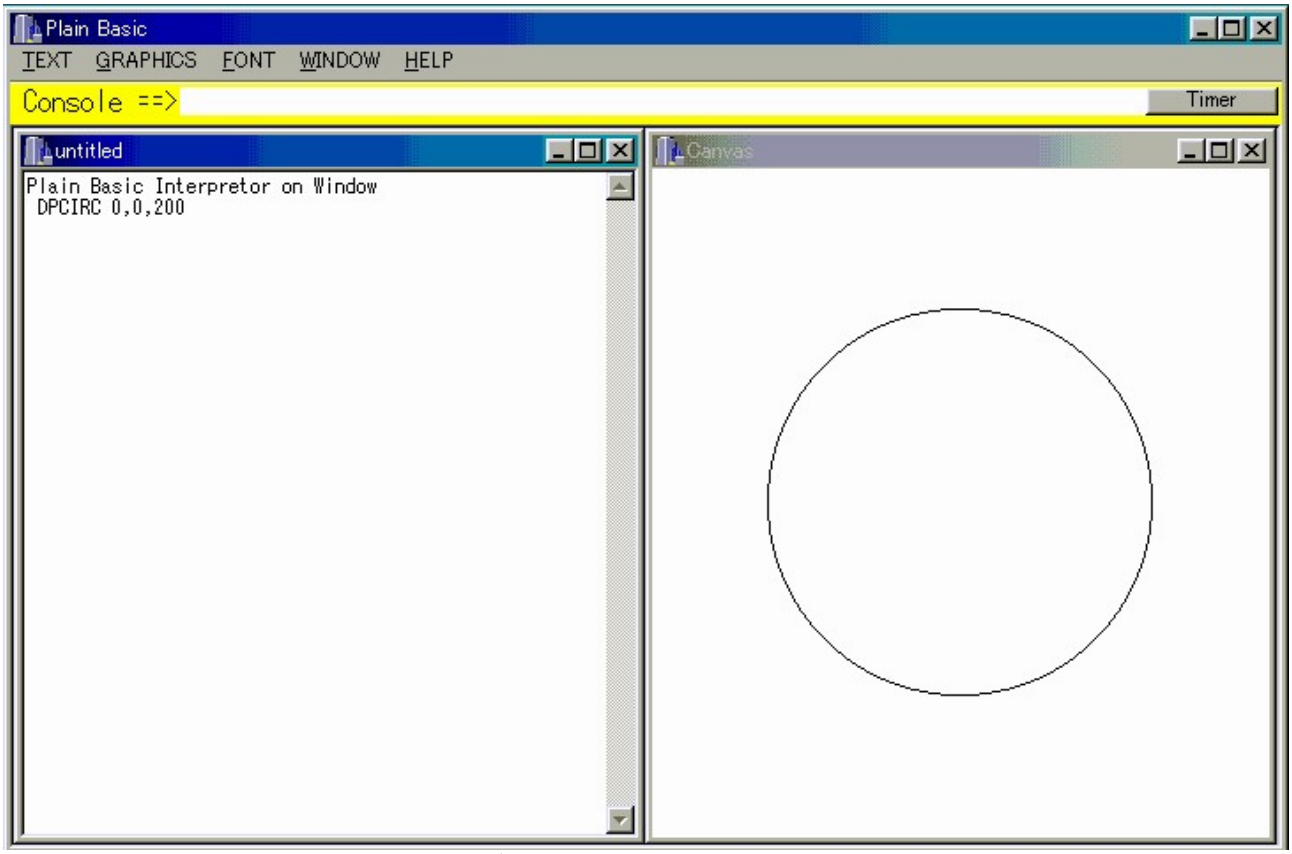

図 2.1　Plain_Basic の立ち上げ時の画面（コマンド DPCIRC で円を一つ描かせました）

８ビットのマイコンをシミュレートしたシステム

Plain_Basic は、Windows のパソコン上で、1980 年代の８ビット**マイコン**の環境に似せた機能を持つプログラムです。８ビットマイコンは、OS と **Basic インタプリタ**が最初から組み込まれた、ハードウェアとソフトウェア一体型のシステムです。Plain_Basic は、32 ビットパソコンと Windows の OS 環境で動作するソフトウェアですが、擬似的に DOS のシステム環境を構成しています。それは、ユーザインタフェースが、原則として CUI にしてあることです。システムと言う用語は組織と訳しています。システムは、幾つかの構成部分が相互に関連を持ちながら全体として機能する集合体を意味しますので、広く使われるようになった用語です。Plain_Basic 主要なシステム機能は、下のようです。

① 代数式を理解できる**関数電卓**のような使い方ができること。
② この場合、多くの変数領域が使えること。ちなみに、電卓ではメモリを一つしか使いません。
③ 計算処理を組み立てるために、簡単な Basic 言語のプログラムを編集して実行できること。
④ 例えば **PRINT** 文のような小さなプログラム文は、コマンドのように単独に使うこともできること。
⑤ Basic 言語の編集を補助するために簡単なテキストエディタの機能を持っていること。
⑥ 処理結果を表示するために、テキストウインドウとグラフィックスウインドウがあること。
⑦ 基本的な操作方法は CUI ですが、**デモンストレーション**に便利な**バッチ処理**もできること。

マイコン用 Basic のインタフェースに多くを学んだこと

1980 年代のマイコンに搭載されていた Basic インタプリタは、初心者がプログラミングを覚えるツールとして取っ付き易いものでした。しかし、８ビットのプロセッサを使う限り、ユーザがコマンドを追加することができませんでした。つまり、専門別の問題を解決するプログラミングには能力不足でした。その解決方法は二つです。一つがビット数の多いパソコンの利用です。16 ビットパソコンの **NEC PC98** は一時代のブームになりました。それは、当時の 16 ビットプロセッサを使う中型コンピュータに匹敵したからです。もう一つは、パソコンを見限って、大型コンピュータを利用する高級言語に乗り換えることです。最も良く使われたのは **Fortran** です。この言語では、多くの**サブルーチン**を**ライブラリ**に構築しておいて、それを組み込む利用ができます。しかし、マイコンのインタフェースに較べると、**ユーザインタフェース**は良くありませんでした。その解決の一つのアイディアは、Fortran のサブルーチンを、あたかもマイコンのコマンド並に使う方法を開発することでした。そこで、Fortran 言語を使って簡単な Basic インタプリタをコーディングし、そこの追加コマンドとして種々のサブルーチンを繋ぎました。Plain_Basic は、Fortran に代えて Borland の C++Builder 5 で書き換えたバージョンです。教育利用を主な目的として、最小限のインタプリタの骨格に、基本的グラフィックスルーチンを加えました。やや専門的な利用のバージョンも別にあります。主要なものは、幾何計算用のツールをまとめた **Geometry_Basic**(G_Basic)、幾何モデリングのサブルーチンライブラリを組み込んだ **GEOMAP+Basic** があります。これらのユーザインタフェースは Plain_Basic と殆ど同じです。

教育利用を考えて仕様を絞り込んであること

Plain_Basic の言語仕様の骨格は、今は無くなりましたが、「**JIS 基本 Basic**(JIS C6207-1982)」にほぼ準拠しています。Plain_Basic は、高速で動作する大きなプログラムの開発と利用が目的ではなくて、**関数電卓**よりも便利に使うことを目的とし、本格的な高級プログラミングを扱うような重いツールではありません。一般に、何かのプログラムを開発したいときは、それぞれに専門としての目的意識があります。高級プログラミング言語を使うとなると、余分に相当な勉強が必要です。初心者がプログラミングを勉強したいときは、プログラミングの目的意識が明確であるとは限りません。また、高級プログラミングを扱うとなると、予備的な知識のハードルが高くて手に余ります。つまり、電卓の利用と高級プログラミング言語の利用との間に大きなギャップがあります。Plain_Basic は、そのギャップを埋める目的に使う**教育利用**を目的としてあります。したがって、あれば便利かな、と思う機能は、意図的に省いてあります。プログラミングの教育では、何を目的としてプログラムを作成するかの目的意識を決め難いところがあります。**数値計算**だけでは受講者の興味をあまり引きませんので、**グラフィックス**が楽しめるように、グラフィックスのコマンドを加えました。それも、モノクロの**線図**だけで図を描くような最小機能に絞ってあって、カラーや塗り潰しのコマンドを入れてありません。

2.2　基本的なキー操作技法

ユーザインタフェースは CUI であること

Plain_Basic は、基本的に**コマンド起動型**(command driven)処理の集合で構成したシステムです。**フォーカス**がテキスト入力枠にあって（**カーソル**が点滅します)、何かの**コマンド**文字列を入力して改行すると、何かの処理が行われ、それが済むと再び入力待ちに戻ります。フォーカスがテキスト入力枠に無ければ、マウスを使ってテキスト入力枠をクリックして**アクティブ**にします。このような使い方を interactive（**対話方式**）と言います。ユーザがコンピュータに話しかける手段が**キーボード**からの文字入力ですので **CUI**(character user interface)と呼ばれます。コマンドは英字名で定義してありますが、大文字・小文字どちらを使っても構いません。歴史的な経過がありますので、この解説ではキーワードをすべて大文字で書いてあります。一方、コンピュータからの応答はテキストウインドウへの文字出力か、グラフィックスで表示します。マウスを使ってメニューを開く、アイコンなどの図柄をクリックしてコンピュータに指示を与えるなどの方法は **GUI**(graphical user interface)ですが、Plain_Basic では最小限にしか採用していません。

メッセージへの応答もキー操作ができること

Plain_Basic の動作中に、システムから**メッセージボックス**や**ダイアログウインドウ**が表示されることがあります。これらの表示には「ＯＫ」「キャンセル」などの表示のあるボタンがあって、それを選択してマウスでクリックするように要求しています。しかし、これも多くの場合、デフォルトでキー入力を受け付けるようになっています。標準的なキーは次のようです。

- 「ＯＫ」　　Enter Key
- 「キャンセル」など　　Esc Key

テキストエディタの操作法を使うこと

Plain_Basic のテキストウインドウは、このままで単純な**テキストエディタ**の機能を持っています。テキストウインドウをアクティブにすれば、Plain_Basic とは独立して、キーボードからの文字入力を編集することができます。**FILE メニュー**には、ファイルとの入出力のサブメニューを持たせてあります。しかし **EDIT メニュー**は、定番である cut,copy,paste 処理のサブメニューを入れてありません。これらは、キーボードショートカットで処理ができますので、意図的に省きました。念の為、その方法を書いておきます。

- すべてを選択(select all)　　ctrl+A
- 切取り(cut)　　ctrl+X
- コピー(copy)　　ctrl+C
- 貼付け(paste)　　ctrl+V
- 取り消し(undo)　　ctrl+Z

テキスト入力枠の文字処理も、上のキーボードショートカットが利きますので、テキストウインドウ上の文字並びを選択して、貼り付ける方法を使うことができます。

ファイルへの入出力は最小限にしてあること

Plain_Basic 流で書いたテキスト形式のプログラムの保存と読み出しは、**LOAD**/**SAVE**/**MERGE** コマンドを使ってファイルを利用します。しかし、それ以外、データ入出力でファイルを使う方法を含めてありません。これに代わる方法として、**内部ファイル**と言う技法を応用しています。これは第４章で解説します。テキストの編集処理をしているとき、選択された文字データはクリップボードを臨時のメモとして使っています。そのため、並列に **NotePad** や **MS-Word** などを開いて、そことの文字データの遣り取りをクリップボード経由で行うことができます。この機能を使えば、ファイルへの入出力メニューを使わなくても済ますことができます。グラフィックスの結果をファイルに保存したいときには、ウインドウの画面全体をビットマップデータとしてクリップボードに書き込むこともできます。これは Windows のシステム機能として準備されていて、キーボードの **PrtScr** を使います。ただし、パソコンの機種によってキーの場所も、また操作方法も違いますので、利用するパソコンごとに確かめる必要があります。クリップボードに描き込んだビットイメージは、グラフィックス処理の別プログラムで読み出して編集し、ファイルに保存できます。少し手間が掛かりますが、Plain_Basic 本体には欲張った機能をあまり追加しないように設計しました。

2.3　ご破算で願いましては

Plain_Basic を初期化する方法

算盤の**読み上げ算**は、最初に「ごわさんで願いましては」と言って零に戻す操作を促します。同じことを電卓では**クリア**と言い、C の字がついたキーが使われます。Plain_Basic の利用では、プログラムを立ち上げた状態が「ごわさん」に相当します。Plain_Basic では、クリアに相当する処理が４つあります。これらは総てコマンド入力で指示します。

(1) NEW:　　変数と Basic 内部プログラムをクリアします。OK と返事のリストが出ます。
(2) CLS:　　clear screen の意味であって、テキストエディタの画面を消去します。
(3) DPERAS　　グラフィックス画面を消去します。
(4) ERASE 引き数で特定の変数や配列を指定してメモリから消去します。

ウインドウの画面は、前の処理結果がそのまま残っていますので、Plain_Basic を立ち上げ時と同じように「サラ」に戻すのは(1)から(3)までの処理をします。(4)は内部プログラムの実行のときに、内部の変数領域を効率的に使うため、不用になった変数領域をクリアするためです。大きな行列を余分に宣言しなければ、通常の利用で使う必要はありません。

Plain_Basic を電卓並に使う

Plain_Basic は、電卓と同じような手順で数値計算ができます。そのユーザインタフェースは、キーボードからの文字並びの入力(CUI)で行いますが、その入力方法が少し違います。電卓で単純な計算をさせるとき、例えば

3.15 × 5 ÷ 7 ＝　　(2.1)

と順にキーを押せば、答の 2.25 が得られます。Plain_Basic では下のように頭にキーワードの **PRINT** をつけて入力します。キーボードには×÷記号がありませんので、代わりに ＊ / を使います。

PRINT 3.15 * 5 / 7　　(Enter↓)　　(2.2)

電卓のイコール記号に相当するのがキーワードの **PRINT** であって、それを頭に付けます。入力した PRINT 文と、その答とが、続けてテキストモニタに表示されます。

関数の計算手順は電卓と異なる

Plain_Basic は基本的な数学関数を使うことができます（表 2.1 参照)。ただし、三角関数では、角度を度で入力し、「PRINT SIN(30)」と入力すると、答の 0.5 が得られます。関数電卓の場合には、キー入力は角度の 30 を先に入力してから、sin 記号のキーを押す順で使います。コンピュータ内部での計算の進め方は、電卓の順が自然です。しかし、数式で表すときにはキーワードの SIN を先に書きますので、計算手順の変更を括弧（ ）で括ってコンピュータに知らせます。引き数を２つ取る **MOD** や **ATN2** を使うときには括弧を使う表現法が必要です。ちなみに、関数電卓で括弧が使える製品はやや高級ですが、コンマを使って引き数を二つ以上並べる方法はありません。

入力データのエコーが得られること

電卓は一過性の計算機と言って、入力と出力の両方のデータは、計算器側で保存しません。式(2.1)を電卓で計算するとき、数値はモニタ画面に表示されます。演算記号を入力して次の数値を入力すると、前の数値表示が消えます。そのため、人の方で入力データを書いたものを準備し、計算手順を覚えておいて、結果を紙に書き写します。Plain_Basic では、式(2.2)で示した入力文字列のすべてはテキスト入力枠で確認できます。改行によって Plain_Basic がそれを取り込むと同時に、テキスト入力枠の文字並びは消去されますが、これはそのままテキストウインドウに書き出されます。これを**エコー(echo)**の機能と言います。エコーは、通信回線を使ってデータの送受信をするとき、相手側が正しく受信したことを確認するため、受信データをそのまま送信側に送り返すことを言います。インタラクティブな環境では、ユーザの入力記録が残せますので、手書きのメモを作らなくて済みます。また、入力ミスがあって入れ直したいときは、その行をコピーしてテキスト入力枠に貼り付け、そこで修正して再度入力させる方法が使えます。エコーはコマンド **ECHOFF/ECHON** を使って書き出しの中止や再開ができます。デフォルトの設定は ECHON です。

2.4　関数電卓より少し賢い使い方

括弧や関数を使った代数式が書けること

代数学は、具体的な**数**の**代**わりに文字を使って四則演算の約束を表して問題を解く数学の一分野です。代数式は、計算の組み立てを表しますので、それを数値計算に応用するときは、文字の場所に具体的な数を代入します。電卓は、ただ一つのメモリがあって、Mの字のついたキーでデータの保存や読み出しができます。しかし、変数記号としてMを生で使うことをしません。Plain_Basic は、代数式の形にした文字並びが理解できます。例えば、下の代数式を計算するとして、x と y との何かの数値を入れて結果を得たいとします。

$$\sqrt{(x-1.2)^2+(y+4.5)^2} \qquad (2.3)$$

最初に、電卓を使って計算を進める手順を考えて下さい。変数に具体的な数値を入れなければなりませんので、メモ用紙に途中経過を書きながら計算します。電卓の１個のメモリを巧みに使えば、賢く計算を進めることができます。ただし、電卓側に、ルートを開く関数(sqrt)が無ければ、上の計算ができません。図 1.4 の電卓にはこれがありますが、簡易な電卓には無い製品もあります。一方、Plain_Basic では、これを一行のプログラム文にまとめることができます。括弧の使い方に注意して下さい。

```
100 PRINT SQR((X-1.2)^2 + (Y+4.5)^2)      (Enter↓)          (2.4)
```

式(2.3)の表し方を(2.4)のように書き直すのが**プログラミング**の実践です。この書き方には、Plain_Basic の流儀（**文法**）が決められていますので、それを守らなければなりません。他のプログラミング言語では別の書き方があります。ここでは、最初の 100 はプログラム文のラベル（見出しのこと）です。**PRINT** はコマンドを表すキーワード、**SQR** はルートに開く関数名、記号 ^ はべき乗を意味します。X と Y とに具体的な数値を代入するときには、代入文形式で下のように書いて、**GOTO 100** と入力すると結果がテキストモニタに書き出されます。例えば、

```
X=20: Y=30: GOTO 100      (Enter↓)                  (2.5)
```

ここで、コロン(:)は、独立した複数のプログラム文(**multi statements**)を一行にまとめるときの区切り記号(**delimiter**)です。別々の行で入力しても構いませんが、行数を節約する表記の方法です。

余りの計算は関数 MOD を使う

例えば「10÷3 の答は 3、余りは 1」のような整数計算は、数学では答を**商**、余りに**剰余**の用語を当てます。ここでの商は、割り算（除算）の答の小数以下を切り捨てる整数除算です。演算子記号に、通常の実数除算記号 / に代えて整数除算記号 ¥を使うことがあります。普通のプログラミング言語では、数の型に実数型と整数型とがありますので、割り算の答を整数型に代入すれば整数の商が得られます。余りの方は単純計算ができませんので、**剰余関数**を準備するようになってきました。キーワード **MOD** はラテン語の **modulo** から来た用語です。電卓は便利なようですが、余りの計算が直接にはできません。余りの計算は、日常的にはお釣の計算があります。設計計算で、或る寸法を一定間隔で分割するときの「寸法余りを求める」などに現われ、需要は案外多いものです。具体的な計算を助ける関数として、Plain_Basic では剰余計算の関数 **MOD** を用意しました。「10÷3 は答 3 余り 1」の計算をしたいときは、整数型の変数、例えば K と J を使って、下のように入力します。

```
K = 10 / 3 :  J = MOD(10, 3) : PRINT K, J      (Enter↓)          (2.6)
```

上の式で、K の代わりに実数型の変数、例えば A,B を使って試してみて下さい。

```
A = 10 / 3 :  B = MOD(10, 3) : PRINT A, B      (Enter↓)          (2.7)
```

このような計算結果の違いは、数の型の違いを理解することと、剰余関数があることで比較計算が得られます。この計算では、**暗黙の型の定義**が応用されています。何も宣言をしなければ、頭文字が I,J,K,L,M,N であるのは整数型、それ以外は実数型の約束にしてあります。これは Fortran 言語の習慣を踏襲したものです。

変数名を使った最初の時点で変数が準備される

Plain_Basic のプログラム文で代数式を書くとき、式(2.4)～(2.7)にあるように、変数名を適当に決めることができます。しかし、高級プログラミング言語では、前もって変数名を**宣言**しておかないとエラーになります。これは厳密さを要求するプログラミングでは一つの常識ですが、鬱陶しいところがありますので、Fortran 言語ではこの約束が緩やかになっていました。マイコン時代の Basic 言語もこの習慣を踏襲していました。同じように、Plain_Basic も、大きなプログラム作成を目的としませんので、厳格さを薄めてあります。数学表記では、変数名に英字一文字を使い、大文字小文字を使い分けることや、イタリック、下付き文字を付けるなどの習慣があります。プログラミングの場合には文字種の選択に制限がありますので、用途が分かるように、説明を兼ねた、**15 字以内の英数字名**を決めて使います。数値が代入された変数はご破算をしなければメモリ上に残っていますので、続けて別の計算に使うことができます。例えば、変数 A を考えて、下のように入力したとします。

```
A = 3.14 * 5 * 5 : PRINT A        (Enter↓)          (2.8)
```

このようにすると、変数 A に 78.5 が保存されています。そこで、例えば、幾つかの計算式をコンマで区切って、続けて下のように入力します；

```
PRINT A+A, A-A, A*A, A/A          (Enter↓)          (2.9)
```

すると、結果は、続けて「 157　 0　 6162.25　 1 」と得られます。

プログラム文を内部保存して再利用ができること

上で説明した式の表現は、キーボードから直接入力して実行させる方法です。この全体をプログラム文にまとめることができます。その書式は、下の例のように、行の頭に整数の**ラベル**（見出し）を付けて入力します。この整数は、0 以外、32767 以下で、昇順であればよいのですが、標準として 10 刻みです。それは二つの行の間に新しい処理手順の文を追加するときの余裕を持たせるためです。

```
10 A = 3.14 * 5 * 5
20 PRINT A                                          (2.10)
30 PRINT A+A, A-A, A*A, A/A
```

このように入力したことを確認するには、コマンドの **LIST** を入力します。すると、上と同じ番号付きのリストが得られます。実行させるには、コマンドの **RUN** を入力します。Plain_Basic は**実行モード**になって、プログラム文を次々に実行し、結果はテキストウインドウに出力されます。プログラム文の終りまで実行すると、**直接モード**に戻ります。

プログラムを外部ファイルに保存して再利用ができること

Plain_Basic のプログラム文はメモリの中に書き込まれていますので、Plain_Basic を終了すると消えてしまいます。そこで、このプログラム文をテキストファイルとして書き出し(**SAVE**)、そのファイルを次回に読み出して(**LOAD**)再利用することができます。この場合にはファイル名を決めなければなりません。例えば下のようにします。ファイル名の拡張子は(.txt) が無難です。Basic の場合には(.BAS) が定番なのですが、Visual Basic の方で使っていますので注意する必要があります。

```
SAVE "test.txt"
LOAD "test.txt"
```

上の使い方は DOS 流の方式です。Windows の機能を利用するため、ファイル名を省いて単純に **SAVE** または **LOAD** だけを入力すると、ファイルを扱うダイアログボックスが現われますので、そのダイアログウインドウでファイル名を決めることができます。LOAD のとき、ファイル名が見つからない場合もダイアログボックスが出るようにしました。読み出し、または書き込みのファイル名は、テキストウインドウのタイトルバーに書きこまれます。図 2.1 では、まだファイルアクセスがありませんので、untitled と表示されています。

2.5 プログラム文の編集と実行

Plain_Basic は四つのモードを持つシステムであること

高級プログラミング言語、例えば C/C++, Visual Basic, Fortran などを使ってプログラミングをする目的は、実行形式プログラム(*.exe)の作成です。その作業は、テキスト形式のソースコードの編集に始まり、コンパイル・リンク・デバッグなどのツールを機能的に組み合わせて処理します。この一連の作業は、プロジェクトと呼ばれ、全体を管理するソフトウェアシステムとして構成され、通称で**統合開発環境**(IDE: integrated development environment)と言います。Plain_Basic は、実行形式のプログラムを作成することが最終目的ではありません。形式的にはプログラムの作成過程を使いますが、すぐに実行させて、その場で何かの結果を得ることが目的です。結果を得るための Plain_Basic の状態設定をモードと言い、これが四つあります。

① **直接モード**： (インタラクティブモードとも言い、基本モードです。)
② **プログラム文編集モード**： (単に編集モードとも言います。AUTO の入力で移行します。)
③ **プログラム実行モード**： (実行モードとも言います。RUN の入力で開始します)
④ **バッチ処理モード**： (DECK コマンドで開始します)

直接モードで入力する文字並びも広義の実行文

Plain_Basic では、ユーザの何かの指示を待っている状態が直接モードです。Windows の用語では**イベント待ち**と言います。イベントは、主に、「キーボードからの文字入力を受け付ける・マウスでクリックする」など、ユーザのコンピュータに対する働きかけです。Plain_Basic は、主に、キーボードからの文字並びの入力に応答するように設計してあります。ただし、その窓口はテキスト入力枠に限ります。そこに入力する文字並びの書式に文法があって、それに合わなければ警告エラーがでます。

(1) 文字の並びの先頭は英字か数字に限ります。記号はエラーの警告が出ます。
(2) 文字並びは、意味のある文字並び（**トークン**）に区切って判断されます。
(3) タブまたはスペースの並びは、一つのスペース扱いで、トークン区切りを助けます。
(4) 英字で始まる場合は英数字名の単語に切り出します。その長さは 15 字以内です。
(5) 英数字名は予約語（キーワード）と比較し、そうでなければ変数名と見なします。
(6) 予約語の場合は、コマンド名またはそれに準じる名前であれば直ぐに何かを実行します。
(7) 変数名で始まる場合は、代入文の書式でなければなりません。これも直ぐに実行します。
(8) 数字で始まるとき、整数値をラベルとし、残りの文字並びをプログラム領域に保存します。

表 2.1 Plain_Basic で用いる予約語

	機能別分類	予　約　語	モード別(*)
1	プログラム編集	AUTO, DELETE, EXIT, LIST, RENUM	コマンド
2	プログラムファイル	LOAD, MERGE, SAVE	コマンド
3	プログラム実行	CONT, NEW, QUIT, RUN	コマンド
		PAUSE	準コマンド
		END, REM, STOP	ステートメント
4	バッチファイル	DECK	コマンド
5	デバッグ機能	ECHON/ECHOFF, TRON/TROFF	準コマンド
6	型の定義	DEFINT, DEFDBL, DEFSTR	準コマンド
7	変数と配列の管理	DIM, ERASE	準コマンド
8	データ入出力文	PRINT	準コマンド
		DATA, READ, RESTORE	ステートメント
9	プログラム実行制御	GOTO	準コマンド
		IF/THEN/ELSE, FOR/TO/NEXT, GOSUB/RETURN	ステートメント
10	組み込み関数	ABS, ATN/ATN2, COS, EXP, LOG, MOD, RND, SGN, SIN, SQR, TAN	式文の右辺値のみ
11	グラフィックス	CLS, DPCIRC, DPDRAW, DPENSZ, DPENTX, DPMARK, DPMOVE, DPTEXT, DPERAS, DPWIND	準コマンド
12	特別機能	OPTION	準コマンド
(*)備考：コマンドは直接モードで使う。ステートメントは実行モードで使う。準コマンドは直接モードと実行モードの両方で使うことを意味します。モードに合わなければエラーが表示されます。			

コマンドとステートメントの区別

Plain_Basic の予約語の一覧は、機能別に分類したものを表 2.1 に示しました。この予約語は、大別してコマンド用とステートメント用に分けて理解します。**コマンド**(command)とは、引き数を持つこともありますが、基本的には単独で機能する小単位のプログラム名です。例えば、**LIST**, **LOAD** が代表的なコマンドです。直接モードで使うのが基本ですので、プログラム文に使うとエラーです。**ステートメント**(statement)は、予約語を含んだ文の形式を持ちますので**プログラム文**と訳すこともあります。一般に、プログラムの実行時（実行モード）には、複数行のステートメントを組みにしないと完全には機能しません。例えば、**FOR** 文、**GOSUB** 文がそうです。**PRINT** 文も、別のステートメントで値を代入した変数を参照しますのでステートメントですが、コマンドとしても使うことができます。これを表 2.1 のモード別分類では準コマンドとしました。Plain_Basic ではコマンドとステートメントの区別がやや曖昧です。一般のコンパイラ型の高級プログラミング言語では、直接モードでの実行がありませんので、コマンドの使い方はありません。

ラベル番号の自動発行の機能を使う方法

プログラム文を編集するには、行の頭に数字ラベルを付けなければなりませんが、このラベルを自動発行するモードが編集モードです。最初にコマンド **AUTO** と入力して改行します。そうすると、プロンプトの表示が「Auto 10 ==>」に変りますので、英字で始まるプログラム文を入力します。改行ごとにラベルの数字が 10 づつ増えていきます。エコーは、数字ラベル付きでテキストウインドウに書き出されます。編集モードから抜け出すときは、キーワードの **EXIT** と入力します。エコーはラベル付きで表示されますが、この行はプログラム領域には保存されません。プロンプトの表示が元の「Console ==>」に戻って、直接モードになったことが分かります。AUTO コマンドはラベル番号の初期値と増分とを引き数で指定できますので、既に内部で作成されたプログラム文と重複しないように設定します。

編集作業を補助するコマンド

プログラム文は、Plain_Basic の内部作業領域に保存されます。それは一続きの 32 KB 長さのテキスト領域です。ここでは、ラベルは２バイトの整数型に直してあり、一行分の文字数も２バイトの整数型で記録する、やや特殊な仕様です。プログラムの大きさは、文字数にして約 30 KB ですが、これは A4 の用紙に詰めると約５ページ程度の分量です。これより長いプログラムを一度に作成できませんが、**MERGE** コマンドを使って擬似的に**オーバーレイプログラミング**を応用することで解決できます。プログラム全体は **LIST** コマンドを使ってテキストウインドウに書き出すことができます。このウインドウはテキストエディタになっていて、スクロールも利きます。プログラム文の追加は、ラベルを頭に付けて直接モードで入力するのですが、これを補助するのが前項で説明した **AUTO** コマンドです。削除コマンド **DELETE** は、複数のラベルのプログラム文を削除するときに使います。プログラム文を追加するとき、同じラベルの文があれば入れ替えます。番号だけを入力すると、もし、同じ番号ラベルの文があれば、削除と同じです。プログラム全体のラベルの体裁を揃えるときは、ラベルの付け替え(**RENUM**)コマンドで制御できます。

プログラムの実行は RUN と GOTO を使い分ける

Plain_Basic のプログラムは、**RUN** と入力します。このコマンドは、変数領域をクリアして、プログラム文の先頭から実行します。**GOTO** を使うときは、制御が入るラベルを指定します。この場合には、既に作成されている変数領域をクリアしませんので、以前の変数値をそのまま使うことができます。例題は第 2.4 節の、(式 2.4) と (式 2.5) の対がそうです。(式 2.5) に代えて **RUN** とすると 0 しか得られません。プログラムの終了は、プログラムの実行が最終行まで処理したときです。途中の行で終了させるときは **END** を使います。プログラムの実行を一時的に止めて、一旦直接モードに戻す命令は **STOP** です。再開は **CONT** コマンドです。**STOP** させた状態で、変数の値を PRINT 文で確かめることができますので、デバッグを助けることができます。この状態でプログラム文に何かの変更をすると実行制御が狂いますので、**CONT** コマンドで実行再開を指示すると can not CONTinue「継続実行ができない」と言うエラーが出ます。

2.6　バッチ処理

CUI の環境でないとバッチ処理が使えない

Plain_Basic は、原則としてユーザインタフェースを CUI で設計しています。キーボードは、DOS の環境では、論理的にはディスクと同じ扱いですので、キーボードからの入力に代えてディスクからの文字並び入力に切り替えて、同じ処理を進めることができます。これを**バッチ処理**と言います。バッチ処理に使うファイルはテキストファイルであって、ユーザがキーボードから入力する文字並びを、そっくりテキストファイルに構成したものです。使い方は、「DECK ″*filename*″ です。この章の最初に説明した（式 2.2）などを、そっくりテキストファイルに作成すると、バッチファイルとして実行させることができます。

表 2.2　直接モードの入力をバッチファイル ″demo1.txt″ にまとめた例

```
PRINT 3.15 * 5 / 7                                  :REM (2.2)
100 PRINT SQR((X-1.2)^2 + (Y+4.5)^2)                :REM (2.4)
X=20: Y=30: GOTO 100                                :REM (2.5)
K = 10 / 3 :  J = MOD(10, 3) : PRINT K, J           :REM (2.6)
A = 10 / 3 :  B = MOD(10, 3) : PRINT A, B           :REM (2.7)
A = 3.14 * 5 * 5 : PRINT A                          :REM (2.8)
PRINT A+A, A-A, A*A, A/A                            :REM (2.9)
```

Plain_Basic のプログラムも、まとめて自動実行をさせて、デモンストレーションができます。例えば、下のようなリストをバッチファイルとして作成して実行できます。

表 2.3　バッチファイル ″demo2.txt″ のテキスト

```
LOAD ″expolygon.txt″
LIST
PAUSE 10
CLS
RUN
PAUSE 10
DECK ″demo2.txt″
```

バッチファイルは入れ子で使うことができる

表 2.3 の例題バッチファイルは、直接モードで「DECK ″demo2.txt″ (enter)↓」で実行を始めます。ファイルにあるプログラム″expolygon.txt″を読み出し、リストを 10 秒間表示して実行します。実行後、10 秒待って、同じバッチファイル″demo2.txt″を読みだしていますので、何回も同じ処理を繰り返します。このような使い方は、デモンストレーションに使うことができます。この無限ループから脱出するには、Esc キーを押します。

教育利用に応用するとき

Plain_Basic の例題は小さなプログラム単位を多く作成することになりますので、全部に眼を通すとなるとキー操作が鬱陶しくなります。そのため、幾つかのデモ用のバッチファイルに例題プログラム名を入れておいて、これをバッチ処理で走らせれば中身がすべて閲覧できます。Plain_Basic をプログラミング教育に使う場合、多くの受講者の成果プログラムを実行させて検査するには手間が掛かります。バッチファイルにまとめておくと、すべての成果を講師側だけでなく、受講者も他の人の作品を連続的に閲覧できる便利さがあります。なお、グラフィックスの例題は、成果を印刷した形で閲覧するように、このマニュアルの後半にまとめました。

3. プログラミングの文法

3.1 プログラミングは一種の英作文である

コンピュータの用語は難しいこと

コンピュータを勉強したいときの難関の一つは、膨大な数の専門用語とそれが意味する実体を理解しなければならないことです。それは、英語・英語らしき言葉・造語・省略語・カタカナ語・日本語・その短縮形などなど、大変です。例えば、personal computer、 PC、 パーソナルコンピュータ、パソコン、とありますが、元は最初の一つです。このように種類が多いのは、人と人とで相互に話し合いをするときに使うからです。コンピュータはアメリカ主導型で開発・発展してきましたので、殆どの用語の原点は英語です。しかし、英語圏であっても、特殊な専門用語がありますし、日常言語も意義の違う使い方をすることがありますので、英語でも専門用語辞典(glossary)があります。日本語の専門用語は、これを翻訳して使うのですが、いつもピッタリとして言葉があるとは限りません。カタカナ語にすることが多く、そうすると字数が増えます。一種の業界用語や隠語のような変質も受けます。パソコンが一つの例です。そのため、日本語でも専門用語辞書が必要です。しかし多くの辞書は、「パソコン：personal computer のこと」で済ますことが多いので、「落馬：馬から落ちること」のようなもので、実体の説明になりません。これを、どのように解きほぐすかが、教育の問題です。

コンピュータに語りかける用語があること

コンピュータは電気・電子的な装置ですので、それを動作させる信号があります。このとき、コンピュータを人間並に扱って、その人に語りかける方法が工夫されました。対話方式(interactive)や、**ユーザインタフェース**(human-machine interface)がその概念を含む用語です。プログラミング言語は、コンピュータに語りかけることを目的として決めた用語です。それは、英語または特殊に変形した英語が多く使われます。そして、日本語の環境であっても、元のスペルで使わなければなりません。プログラミングを覚えることは、このプログラミング言語の使い方を覚えることです。プログラミング言語は、英語の方言(dialect)のような性格があります。そしてプログラミングは、この英語の方言を使った英作文の趣があります。したがって、プログラミングに入る前に、日本語で、何をしたいかの目的意識を持ち、それを論理的に組み立てておいて和文英訳、つまり、プログラミングに取りかかります。

最初からコンピュータ側が理解している用語が予約語である

英作文をするとなると、単語と文法の知識が必須です。プログラミングの場合には、必須の単語の種類は多くありません。これが予約語、またはキーワードです。これを補うため、ユーザ側で自分の都合のよい変数名や処理名を決める自由度があります。第三者が作成した処理を流用したいときは、そちらで決めた名前を使います。そうすると、見掛け上、覚えなければならない単語の数が一挙に増えます。初心者が、例えば高級プログラミングのＣ言語を覚えようとするときに、標準ライブラリの関数名に一通りの知識が必要になること、隠されている別ファイルに用語が宣言してあることが、理解を難しくしている一つの理由です。Plain_Basic では、表 2.1 で示したように予約語の数が 60 程度ですし、余分なライブラリを使いませんので、全部を覚えるとしても簡単です。Plain_Basic では、予約語の全リストを HELP メニューの Keyword List を使うとテキストウインドウ画面に表示します。また同じく Manual を引けば HtmlHelp で説明を見ることができます。

文法は記号と数字の使い方も含むこと

プログラムの文構造は、コンピュータを擬人化し、その人に対する**命令文**の形を多く使います。この構文は、動詞が文頭に来ます。名詞を使うこともありますが、「…をしなさい」の意義で使うと考えることができます。英語の文法を覚えるとき、案外見過ごしている規則は、コンマやピリオドなど、記号の使い方です。式を表すのは**式文**(expression)と言って、記号を含む数学式を指します。しかし、英語の環境では、算法そのものも、文章で表すことが元にあって、それを記号化したのが式です。例えば、「A=B+C」は、「Let A be equal to B plus C.」を記号化したものです。少し古典的ですが、文頭に、動詞にあたる LET や COMPUTE を付ける場合もあります。日本語では、明確な命令形がありませんし、動詞が文末に来る語順ですので、この言語習慣の違いがプログラミングに微妙に影響します。コンピュータは、物理的な文字並びを論理的な意味並びに直す処理をしますので、記号を含む文字並びの約束をしっかりと決め、人の方はこの約束に従ってプログラム文を書きます。この約束が文法です。

3.2　英文は語順の習慣があること

直接目的語が文末に来ること

英語と日本語との大きな違いの一つは、語順の相違です。英語は、日本語の助詞（がのにを）に当たる語がありませんので、単語の並べ方で意味が確定します。それは、主語(Subject)・動詞(Verb)・目的語(Object)の順ですので、SVO と略記します。日本語の標準は SOV です。プログラミング言語では命令文を使いますので、VO の形式が多くなります。他動詞は目的語を二つ使うことがあって、英語では間接目的（…に）・直接目的（…を）の順に言葉を並べ、直接目的を文末に置くのが標準です。日本語では助詞が使えますので語順に自由度がありますが、（…を…に）の順が普通です。日本語は「餌を猫にやる」のが普通の語順です。その理由は、「餌」に注目すると、文の並びが「餌→猫」の動きと一致するからです。英語の語順は「やる、猫、餌」になって「猫←餌」の逆順の表示になります。

英語では割り算の表現が二通りある

日本語の環境では、A=B/C では「B を C で割る」と言い、直接目的 B を先に言うのが普通です。英語では、前置詞の違いによって「B divide by C」と、「C divide into B」の二通りの言い方があります。後者の言い方は日本語の習慣にありませんので、かなりの人は知りません。しかし、事務処理用の言語 Cobol では、この二種の表記を許しています。

代数式表現も処理順から見れば逆順

代数式「A=B+C」は、数学表記で見れば等式と言い、「B+C=A」と書いても意味は同じです。しかし、プログラム文の構成では、後者の書き方を許していません。処理の動作で見れば、前者の表現は「B+C→A」の動作を表します。電卓の動作も、またコンピュータ内部でもこの順で計算が実行されますので、プログラム文をコンピュータが解読するときは、内部で手順を変更します。そのため、この代数式は、プログラムの構文では論理的には二つに分けます。右辺の「B+C」の部分を**式文**、イコール記号＝で繋ぐ形式を**代入文**(assignment)と言います。そして、A をイコール記号の**左辺値**(lv: left value)、右を**右辺値**(rv: right value)と言うことがあります。代入文の内部処理は、「lv←rv」の逆順です。

式文の処理は優先順位があること

加減乗除の記号(+ - * /)を挿んで代数式を書くとき、例えば「A+B*C-D/E」とあれば、原則として左から右の順に計算を進めるのですが、掛け算と割り算の部分を先に計算して、実際には「A+(B*C)-(D/E)」と解釈して計算します。電卓を使って「1＋2×3－4÷5」と入力すると、答は 1 です。Plain_Basic では、「PRINT 1 + 2 * 3 - 4 / 5 (enter↓)」とすると、答は 6.2 になります。これは、掛け算・割り算を先に計算をしておく数学上の約束を使うからです。これを、「掛け算・割り算は足し算・割り算より優先順位が高い」と言います。最も優先順位の高いのは括弧( )で区切った内側の計算です。優先順位があるときは、計算の途中結果を保存しておく必要があります。電卓は括弧が使えませんし、演算子の優先順位もありません。内部に三つのレジスタがありますので、途中の計算結果を別のレジスタに移すような、簡単なことしかしません。

べき乗の計算は記号と優先順位に注意が必要

多くのプログラミング言語では、演算子の優先順位が決められています。この中で、べき乗記号を使うときの優先順位の約束が、言語によって異なりますので注意が必要です。Plain_Basic では、べき乗記号に ^ を使い、乗除記号 * / と同順位です。例えば下のように計算されます。

2*3^2 →36、　　2*(3^2) →18、　　2^3*2 →16、　　2^(3*2) →64、

科学技術関係で使う数式では、べき乗計算が最も優先順位が高い約束です。Fortran はこの約束を使いますので、上の例では最初の二つは同じ 18 です。BASIC 系の言語も以前はこの約束でしたが、Visual Basic ではべき乗・掛け算・割り算は同順位になりました。C/C++言語では、べき乗は関数 pow()で計算し、演算記号を使いませんが、括弧の中を先に計算しますので、優先順位が実質的に高くなっています。なお、昔の Fortran の言語では演算子記号に ^ が使えませんでしたので、**と繋いだ記号をべき乗記号に使いました。

3.3　非実行文と実行文

定義と宣言は非実行文とする

プログラムの構文を説明するとき、その文がする処理について、**非実行文**と**実行文**とに分けます。これは、プログラマ側、つまりユーザ側に説明するときの便宜的な分類であって、コンピュータ側ではどちらも何らかの処理を伴います。プログラミングはコンピュータに伝える文書を作成する作文作業ですので、最初に約束事をコンピュータに伝えます。これが非実行文であって、大部分は定義と宣言です。表 2.1 に示した Plain_Basic の予約語では、5.デバッグ機能、6.型の定義、7.変数と配列の管理、8.データ入出力文の中の DATA、がそうです。残りが実行文に使う予約語です。あらかじめ定数などを決めておくのは、Plain_Basic では代入文で行いますが、高級言語では定義と宣言で扱うことも行われます。

定義と宣言の区別を理解する

プログラミングの参考書を見ると、定義と宣言の用語が随所に出てきます。しかし、その使い分けは、かなり曖昧です。**定義**(definition)と言うのは約束事を指します。専門用語では**仕様**(specification)も使います。Plain_Basic の予約語は、そのスペルを決めることと同時に、語の使い方を定義します。逆に言えば、予約語にない語を使っても何もしませんし、最初に決められた機能以外を使うこともできません。典型的な定義は、変数の型の定義です。Plain_Basic では、**整数型**・**実数型**・**文字型**が定義されています。他のプログラミング言語にある論理型・複素数型などはありません。また、整数型・実数型ともに内部的には 8 バイトの倍精度実数型が使われていますが、整数型は小数以下を切り捨てています。また文字型は 15 バイトまでの文字列を保存すると約束します。これが定義です。その約束の拡張として、型の定義をする DEFINT/DEFDBL/DEFSTR があって、英数字変数名の頭文字の役目をユーザ側で変更することができます。定義の段階では、まだ具体的な変数が準備されていません。そこで実体を持つ変数は、**宣言**(declaraion)して決めます。この方法は高級プログラミングでは必須の処理ですが、Plain_Basic では、DIM 文で配列を宣言するときだけです。一般の変数は、代入文などで最初に変数が現われたところで宣言を兼ねて変数の実体が作られます。そのため、実体を削除する命令もあります。ERASE がそうです。

読み飛ばしの指定をコメントとして使う

一般に、プログラム文の文字並びでは、コンピュータが読み飛ばして、何もしないことを指示する方法が決められていて、それをプログラマのメモやコメントの挿入に使います。文字並びも、二つ以上続くスペースやタブを一つのスペース扱いをしますので、原理的には文字の読み飛ばしです。スペースは、プログラム文のスタイルを調整して、人の方で読み易い書類デザインにすることに応用します。Plain_Basic の予約語 REM は、remark の意味を持たせたスペルであって、その語以降、行末までを読み飛ばすようにコンピュータに指示します。分類としては非実行文に含ませます。専門的になりますが、コンピュータの機械語レベルには、コンピュータが本当に何もしない命令語(NOP)があって、機械語命令の余白を埋める目的に使っています。

実行文の主役は計算であること

実行文を分類する方法は幾つか提案されています。名称としては、**代入文**・**入出力文**・**制御文**があります。制御文の細分類は、**繰り返し文**・**条件文**・**飛び越し文**に分けます。プログラムの動作から言うと、代入文と入出力文はデータの移動を扱う処理、制御文は実行位置の移動を指示する文です。コンピュータは計算の道具ですので、計算部分が主役であって、その場所が CPU 本体です。Plain_Basic が関数電卓を便利にしたツールに位置付けるように設計した理由は、これがコンピュータ利用の原点だからです。CPU にデータを送り、そこからデータを得る処理を巡って、多くの文を組み合わせます。計算部分は記号並びで表すことから、文扱いをしませんが、**式文**の用語があります。式と言うと**数学式**だけを想像しますが、**論理式**もあります。後者の方は初心者が面食らう計算法です。データの移動のとき、データの加工や変換を行うことがあります。数字から文字に変える変換、または文字から数字に変える変換は入出力文の時に応用します。しかし、これはプログラミング言語では最も手間の掛かるところです。Plain_Basic では、入出力文を最小限に抑えていて、PRINT 文もデフォルトの書式変換を使います。

3.4　制御文の使い方__その１：FOR-NEXT 文

順に実行させる筋書きがプログラム

プログラムは一般的な言葉ですので、運動会のプログラム、音楽会のプログラムなどと使われます。したがって、コンピュータで使う場合には、厳密に言うときはコンピュータプログラムと断ります。コンピュータがプログラム文を解読するときは、最初の文から読み、人が文書を読むような先読みをしません。拾い読みもしません。しかし、脱線して別のページを参考にすることもありますが、元に戻るのが基本です。脱線、つまり処理の流れを意図的に変えるのが制御文です。その定番は、FOR 文、IF 文、GOTO 文、の三つです。

繰り返し計算ができるプログラミング

コンピュータが電卓よりも便利に利用できる利点の一つが繰り返し計算のプログラミングができることです。電卓には繰り返し計算の機能がありませんので、一過性の計算機とも言います。BASIC 言語の定番は、キーワードとして、「FOR-TO-STEP-NEXT」の組を使います。他のプログラミング言語でも同じような機能があります。キーワードや文の書き方が違いますが、本質的には同じです。C 言語では「for(; ;) { };」の形、Fortran では 「DO-CONTINUE」の対で指定します。FOR-NEXT の使い方は、下のような単純なプログラム例を試すことから始めるとよいでしょう。省略した書き方もありますが、入門としては、下の例のようにキーワードを総て含めた表示法で覚えます。

```
10 FOR K=1 TO 10 STEP 1
20 PRINT K
30 NEXT K
```

RUN で実行すると、コンソールには、1 から 10 までの数が書き出されます。ラベル 10 の文で、数字を種々に変えて結果を試すと納得できるでしょう。この繰り返し文のループを使うときの注意は、何回ループを回るかの回数と、ループを抜けた後の数がどうなっているかです。

無限ループが怖いこと

プログラミングで、FOR-NEXT 文を使うとき、条件が合わないと FOR-NEXT ループから抜けられなくなります。そうすると、キーボードやマウスの制御を受けつけなくなります。これを「無限ループに入った」、「コンピュータが暴走している」、「ハングアップ」している、などの症候を示します。プログラマは、このようにならないように注意します。この条件は、例えば、上のステートメントで、例えば、「K=1 TO -10 STEP -1」と書くところを、STEP 以降を書かないと起こります。条件として K=10 が成立しませんので暴走が起こります。Plain_Basic は教育利用を意識して作成しましたので、繰り返し回数が 30000 回を超えるような初期条件は、エラーを出すようにしました。また、Plain_Basic の FOR-NEXT ループでは、ループの途中からループを抜け出る方法や、カラ回りの指定には GOTO 文を使うことができます。しかし、これは「プログラミング道」から言うと、一種の禁止事項です。高級プログラミング言語では、構造化プログラミングを意識して GOTO 文を使いませんので、例えば C 言語では、break 文や continue 文が準備されています。

高級言語では While 文もある

繰り返し計算は、何かの終了条件を判定して繰り返しを中止しなければなりません。その判定は、原理的には、後の節で説明する IF 文の条件式を使います。FOR-NEXT 文は、この判定に媒介変数の大小比較を使いますので、他の条件を使って繰り返しを停止したいときに困ります。例えば、数の並びを順々に見ていって、特定の数が見つかったところで、その位置を持ってループを出たいとします。それは GOTO 文を使うことになるのですが、そうすると NEXT 文の約束が破られますので、次に別の FOR-NEXT 文に入ると、おかしな結果になります。そうかといって、FOR-NEXT 文の最後まで付き合っていたのでは無駄にループを回ります。そのため、条件文で繰り返し計算を抜ける方法が必要です。Plain_Basic では使いませんが、良く使われるキーワードが while です。C 言語では一種類ですが、Visual Basic には種々の変形があります。単純な形は、While, End While の対で使います。Do, Until, Loop のキーワードを組み合わせた制御方法もあります。

3.5　制御文の使い方__その２：GOTO 文と GOSUB-RETURN 文

プログラム文にラベルが必要であること

高級プログラミング言語では、**GOTO** 文を殆ど使わないようになりました。GOTO 文は便利なのですが、多用するとプログラム文を論理的に追いかけることが難しくなって、いわゆる**スパゲッティコード**になる、と言う理由からです。しかし、初心者が小さなプログラムを作成する程度であれば、GOTO 文は処理の流れを理解する助けになります。GOTO 文を使う場合には、飛び込み先に目印が必要です。これが**ラベル**です。Plain_Basic では、古典的な方法ですが、すべてのプログラム行に整数で昇順のラベルを付けます。Fortran は、GOTO 文を陰に陽に使い分けますが、必要である箇所に数字のラベルを付けます。この数字は昇順でなくても良いことになっています。マイクロソフト社の Visual_Basic では、ラベルが必要である箇所の行頭に、英字名か数字のラベルを付け、コロンで区切る使い方ができます。

初心者教育には GOTO 文の理解が必要

GOTO 文は、プログラマレベルの構文では悪役なみに扱われるようになりましたが、機械語レベルではジャンプ命令と言い、機械語のコード並びではジャンプ命令のオンパレードです。GOTO 文を避けるプログラミング技法は、まとまった処理単位をサブルーチンや関数にまとめ、名前を付けて参照する方法が応用されます。これらがライブラリになっていると中身を見ることができませんので、プログラマにとってはブラックボックスになってしまいます。GOTO 文は飛び込み先が見えています。このような得失がありますので、プログラミングの初心者教育では GOTO 文を積極的に使うのが良いでしょう。

GOTO 文を使うときの注意

GOTO 文は、決められたプログラム文の順番を無視して飛び出し、別のところに飛び込みます。FOR-NEXT で囲まれたループ枠から外に飛び出したり、また、その中に飛び込んだりすると、NEXT の制御条件が狂いますので、なにが起こるかの予測が付きません。そのため、最初から、飛び出し・飛び込みを避けるか、飛び出しても元に戻るようにプログラミングを工夫します。このことを保証する制御文は、次に説明する GOSUB-RETURN の対です。また、重複する FOR-NEXT 文は、入れ子構造に多重化するように使います。プログラム文単位では**文法的**に(syntactic)正しくても、文の組み合わせで**意味的**な(semantics)使い方に誤りがないようにプログラミングをしなければなりません。Plain_Basic は文法エラーにはメッセージを出すようにしてありますが、論理的なエラーを判断する機能はありません。高級言語では、部分的ですが論理的な構文エラーにも警告を出してくれます。

GOSUB-RETURN 文は小さな処理単位をサブルーチン扱いにする

高級プログラミング言語を使って全体で大きなプログラムを作成するときは、幾つかの小単位のプログラム単位を繋ぐようにすると管理し易くなります。この単位のことを**サブルーチン**や**ファンクション**などにまとめます。これらを個別にファイルにまとめるのも、また、全体を大きな一続きのファイルに詰め込むのも扱い難くなりますので、幾つかの単位をまとめてファイル化します。Plain_Basic は教育利用を主な目的としますので、最初から一続きでプログラム文の集合を書き、これ全体を一つのファイルにします。ファイルの寸法も、約 30KB 以内に制限しました。この内部に、サブルーチンと同じような機能を持つ独立した小単位ルーチンを含ませることができます。これを利用する方法は、呼び出したい場所で、「GOSUB 2000」のように飛び込み先のラベルを指定します。そこでは、一区切りの処理が済む場所に RETURN 文を入れます。そうすると、制御は GOSUB の呼び出し位置の次に帰って、処理が続行されます。飛び込み先は、整数ラベルだけで、特別な見出しを使いませんので、コメントなどで分かるようにします。この扱い方に慣れると、高級言語でのプログラミングでの、サブルーチン、モジュール、プロシージャなどを組上げるときに、考え方の応用が利きます。Fortran ではサブルーチンを呼び出すときに CALL subname の書式を使います。Visual Basic も Call 文が使えます。使わない書式もサポートされています。C/C++言語では、CALL などのキーワード無しに直接サブルーチン名を呼ぶ書式です。

3.6　制御文の使い方__その３：IF-THEN-ELSE 文

二つの内どちらかを選ぶ処理が基本であること

プログラムの実行位置を一意に変えるのはGOTO文ですが、何かの条件を判断して、「右か・左か」のように、二つある選択肢のどちらかを選択して実行位置を変更する方法が必要です。これには「IF condition THEN statement-1 ELSE statement-2」の構文（IF 文）を使います。Plain_Basic では、プログラム文の一行文字数を多く許していませんので、条件(condition)と二つのステートメント(statement-1/-2)が長いと、一行に納まらないことも起こります。一行の文字数を節約する方法の一つは、statement に GOTO 文を使って次に実行するプログラム文の入口を指定します。第二は、条件(condition)が成立しないとき(false)の処理をデフォルトとしておけば、ELSE以降を使わなくても済みます。三つ以上の選択肢があるときは、Plain_Basic では、面倒でも IF-THEN-ELSE 文を二つ以上組み合わせます。高級なプログラミング言語では、多重な条件が処理できる文法がありますが、その基本として単独の IF-THEN-ELSE 文を省略なしに使う方法を理解しておくことが大切です。数値計算では論理判断があまり注目されませんが、これが理解できないと実践的な計算が組み立てできません。

論理学の基礎知識が必要であること

日常言語では、例えば「雨が降ったら家に居て、そうでなければ釣に行く」のような言い方があります。コンピュータで応用する IF 文は、この条件判断の文章表現に記号を使う論理式を使います。この体系を記号論理学と言います。この解説だけでも多くのページ数が必要ですが、ここでは省きます。記号化するとき、「雨が降ったら」が condition です。俗に言う「れば・たら」の条件です。このところに使う式は論理式と解釈されて、計算結果は「**真**(true)か**偽**(false)のどちらか」です。同じ条件判断で、「雨が降らなかったら釣に行き、そうでなければ家に居る」と書き直すことができます。日本語の習慣では、「れば・たら」の部分が論理的に正しいときに「はい」と言いますが、英語では返事の肯定否定で yes/no を使い分けますので、この言い方を避けて、true/false の用語を当てます。つまり、IF 文の論理の帰趨は、日本語の感覚の方が素直で分かり易いのです。

ド・モルガンの法則の実際

二つ以上の条件の重なり、例えば、「変数 K が 3 以上 10 以下であれば」という条件を論理式で書くと

```
IF （ (K >= 3) & (K <= 10) ） THEN statement-A ELSE statement-B
```

同じことを下のように書き換えることができます。これが**ド・モルガンの法則**の応用です。

```
IF （ (K ＜ 3) | (K ＞ 10) ） THEN statement-B ELSE statement-A
```

文章の例で説明すると「雨が降って気温が低ければ…」と「雨が降らなくて気温が低くなければ…」と言い換えることを法則としたものです。この論理式は二つの条件文を組み合わせた形になっていて、&(AND)と|(OR)は、それを繋ぐ論理演算子です。込み入った条件文を使うときは、AND, OR, NOT, XOR, IMP のような、文字を使う演算子を定義することがあります。しかし、この演算子を使わなくても、少し泥臭く見えますが、IF文の繰り返しで同じ判断が処理できます。８ビットのマイコンで使ったBASICには、AND, OR などの論理演算子がありませんでした。Plain_Basic も基本的なバージョンには、文字で表す演算子を含ませていません。上のような表現方法ができると、プログラム文がスマートに見えます。**ド・モルガンの法則**は高校の数学で扱っています。プログラミングで、この法則を巧みに応用できるようになれば、IF 文の勉強は卒業です。

4. ファイルを使う処理

4.1 ファイルとディスクの用語

ファイルの用語は事務処理から転用されたこと

古い時代のコンピュータは、プログラムのソースコードやデータをカードやテープなどの**媒体**で扱いました。この媒体を、古い用語では DECK と言いました。データの中身を言うときは**データセット**(data set)と言いました。事務処理用のプログラムを作成するとき、扱う中身は帳票など、紙に書いたデータですので、データの集まりを**ファイル**と呼ぶようになりました。**フォルダ**も文房具の用語ですが、ファイルの集合をそう呼ぶようになったのも自然です。紙に代わる媒体を扱う装置を**ファイル装置**と呼ぶようになりました。媒体は、紙テープ・磁気テープ・パンチカード・ディスクなど、種々の開発がされました。テープレコーダのような装置もありますが、円盤(ディスク)を使う方法が一般化しましたので、ファイル装置よりもディスク装置の呼び名が一般化しました。媒体が変れば、それを読み書きする装置が変りますので、プログラムも書き換えなければなりませんでした。そこで、データの中身が同じならば、装置の相違があっても、プログラムの変更がないような工夫が必要でした。そこで登場するのが**オペレーティングシステム**(OS)であって、装置の相違をここで吸収するプログラムです。最も利用頻度の高い媒体が**ディスク**であることから、他の総ての入出力装置も、抽象化して**ディスク装置**として扱う OS が工夫されました。これが **DOS**(disk operating system)です。ディスク装置の広い概念には、キーボード、モニタ、プリンタも含みますので、初心者は面くらいます。DOS は一般名詞扱いの用語です。MS-DOS,PC-DOS などは商品名であって固有名詞扱いです。

ファイルの扱いはオブジェクト指向であること

ファイル装置は典型的なハードウェアですので、それを扱うプログラミングは、新しい用語で言えば**オブジェクト指向プログラミング**(object oriented programming)です。オブジェクト(object)は日本語に訳し難い用語ですので、カタカナ語で使います。そのため、良く分からなくて、誤解も起こります。文法用語では**目的語**と訳していますが、これは名訳です。それに引きずられて「目的」と解釈すると誤解が起こります。オブジェクトの単純な意味は、形を持った物（もの）であって、味も素っ気もありません。英語の環境では単純な用語ですが、日本語の環境で意味を探ろうとすると面倒な専門的な解釈が入ります。注意することは、文法用語の目的語が意味する対象は、物と同時に人や生き物も含むことです。コンピュータが処理するのは眼に見えない電気信号ですし、数値計算も実体がよく分からない処理です。これに対して、それらを扱う、眼に見える物をオブジェクトの名称でくくります。ディスクは典型的な物ですので、ディスクを扱うプログラミングはオブジェクト指向プログラミングです。その手続きは、準備作業(例えば open)、入出力処理(例えば read/write)、後始末処理(例えば close)の三段階あるのが定石です。ファイル作業は、どの装置を使うか、ファイル名は何か、読み・書きどちらか、などを決めます。これらの情報を一括管理するデータ領域が内部的に準備されます。新しい概念では、この全体を**クラス**として扱います。ユーザレベルでは、その中身について詳しく知る必要がありませんので、DOS の環境のプログラミング言語では簡単になっています。例えば、キーボード、モニタ、プリンタなどは広い意義ではディスク扱いですが、準備作業も後始末処理も表には現われません。しかし、Windows の環境では、これもユーザの責任でオブジェクト指向プログラミングをしなければなりません。

電卓はディスクを使用しないコンピュータ

コンピュータは、ディスク装置無しでは使いものになりません。ディスクおよびディスク装置の取り扱いは、注意深さが必要ですので、昔は一般ユーザに触らせないように管理しました。媒体としてのフロッピーディスクは、ファイルの扱いを大衆化しました。しかし、それまでの経過がありますので、ファイルを扱うプログラミングは、初心者には難しいところがあります。マイコン BASIC は、ファイルの入出力に関係するコマンドが当初はありませんでした。ディスク装置は精密な機械装置ですので、初心者にファイルを扱わせるのは危険だからです。それではプログラムの保存などが不便ですので、ディスクが使える BASIC を搭載したマイコンが発売されました。その場合の最小限のコマンドが **LOAD/SAVE/MERGE** です。当初は、これだけで用が足せました。ファイルは、大量のデータ処理をするときには必須ですが、Plain_Basic は教育利用を主目的としますので、意図的にデータファイル関係のステートメントを省きました。例えば、**OPEN/CLOSE**、**INPUT#/PRINT#**はありません。これでは不便ですので、次節で説明する内部ファイルの取り扱いを含ませてあります。

4.2 内部ファイルの概念

ディスクはコンピュータの付属品であること

一般に、コンピュータプログラムは、プログラム本体が一つで、外部装置であるディスクにデータを個別にファイル単位で作成しておいて、ファイルを選択して処理します。したがって、プログラムから見れば、ファイルはプログラムの外にあります。しかし、特に外部ファイルとは言いません。Plain_Basic は、一つの実行形式のプログラムですので、Plain_Basic から見れば、BASIC 文で書いたプログラム文がデータに当たります。複数の BASIC プログラムファイルを作成し、それを選択して読み込み、実行させます。その BASIC プログラムが、個別に処理用のデータファイルを必要とすると、ファイルの種類が増えますので、どの BASIC プログラムがどのデータファイルを使うかが、ファイル名だけでは分からなくなることが起こります。学校教育で利用すると、受講者の数が多くなりますので、この問題が発生します。これを避ける一つのアイディアが内部ファイルと言う概念です。これは、プログラム文とデータ文とを一つのファイルにまとめる方法です。データ文の方は、プログラム文から言えばファイルの扱いですが、プログラム文と同居していますので、内部ファイルと言います。

キーワード DATA を頭に付けて内部ファイルに構成する

Plain_Basic は、BASIC プログラム本文とデータとを一つのテキストファイルに作成します。どちらも整数ラベルを付けたテキスト形式のプログラム文ですが、データの場合には、データであると宣言するキーワード **DATA** を行頭に付けます。プログラムの実行時には、DATA 文は読み飛ばされます。**READ** 文が内部ファイルを読み出すステートメントです。変数名リストを解読した後で、内部ファイル読み込みのルーチンが起動します。こんどは、キーワード DATA の付いた行だけを選択して文字並びを読み込み、それを解読して変数に代入します。DATA 文の行は、順番に注意すればどこにあっても良いので、BASIC 文ソースコード全体の頭や末尾にまとめます。これと機能的には同じことを Fortran 言語でも利用しました。プログラム本体に組み込んだ文字列データをファイル並に扱って変数に読み込みます。その命令語はファイル読み込みと同じ書式の READ 文です。メモリ内に作成したテキストデータをファイルデータとして扱うことから**内部ファイル**と呼びます。マイコン BASIC の言語仕様の READ 文はこれを受けています。高級なプログラミング言語では、内部ファイルの考え方を使わなくなりました。その代わりに外部ファイルの扱いが充実しています。

擬似的にデータファイルの読み込みができる

Plain_Basic では、BASIC プログラム文の実行時に、データを外部ファイルから読み込むステートメントがありません。それを補う方法として、MERGE コマンドを使って、プログラムファイルを継ぎ足す、または前にあったプログラム文と差し換えます。このプログラム文が DATA を付けたステートメントであれば、データファイルを読み込む方法と同じ処理ができます。なお、ファイルへの書き出しもファイルの扱いですが、Plain_Basic では PRINT 文を使ってコンソールへの書き出します。これから、間接的に外部ファイルに書き出すことができますので、直接に外部ファイルへの書き出し用のステートメントを省きました。

プログラムとデータの編集

現在のパソコンの環境では、ユーザレベルで、ソースプログラムを読み込んで、それに修正を加えて実行することは殆どありません。そのため、Plain_Basic にあるような **LOAD/MERGE** 命令はなく、完成して変更のできないプログラムを読み込んで実行させます。実行形式になったファイル名を指定することは、Plain_Basic の環境では **LOAD** と **RUN** のコマンドの対に当たります。Plain_Basic は、初心者のプログラミング教育を意識していますので、プログラム本体もデータも、テキストデータとして編集することができて、すぐに実行ができます。その具合を見て、何度でも修正を加えて試すことができます。内部ファイルの扱いを習得していれば、外部ファイルを扱う場合に、殆ど同じデータ構造で設計することができます。実務に利用するプログラムは、プログラム本体やデータが簡単に変更できると、安全管理(security)の面で問題を起こします。しかし、この試行の段階はバグの発見に役立ちますし、種々の知恵を試すことができます。

4.3 便利と危険の両面

ファイルを扱うプログラミングは上書きの危険を伴う

パソコンは、個人が独り占めで使うコンピュータの意義で使うようになった用語ですので、共同利用を考えた大型コンピュータの対義語です。したがって、パソコンではファイルの管理は自己責任が原則です。共同利用の環境にあるときは、他人のディスクファイルを自由にアクセスすることの制限ができます。この措置はパソコンの環境では扱いません。パソコンは、ユーザレベルで自由にファイルを扱えますが、同時に、大事なファイルを消してしまう危険も背負います。DOS の環境では、見掛け上の並列処理ができませんので、何かのプログラムを実行しているとき、ファイル名の一覧を見ることができません。それに対して、Windows の環境では、ファイルエクスプローラを並列に起動してファイル名を探すことができますし、ファイルのダイアログボックスを表示させてファイルを選択することもできます。そこでファイル名を確認すれば、上書きのミスを防ぐことができます。カセットテープには、書き込み禁止にするために、小窓にある爪を折ります。フロッピーディスクや MO には書き込み禁止の窓を閉めると記録ができます。この処置はユーザには分かり易い物理的な安全対策です。しかし、内蔵のハードディスクでは、ソフトウェアで書き込みの禁止や解除をしますので、うっかり同じ名前で上書きや消去をしてしまうことも起こります。始末の悪いのは、インターネット経由で侵入するコンピュータビールスです。内蔵ハードウェアは、現在までのところ、ユーザレベルで物理的な読み書き禁止ができませんので、侵入者に悪用されます。

装置が変れば前のファイルが全滅する危険がある

大量のデータを読み書きできるファイル装置は、形態は異なりますが、嵩張った紙の書類を相対的に小さな記録媒体に圧縮保存します。そのため、データの物理的な保存スペースの節約になります。磁気テープやフロッピーディスクなどの記録媒体は、ファイル装置に依存しますので、異なった仕様のファイル装置では読めなくなり、結果としてその記録は全滅の憂き目を見ます。コンピュータ技術は、表面の華やかさの裏に、使えなくなった装置や読めなくなった記録媒体が死屍累々と横たわっているのです。保守的に見えるようでも、紙の資料は何とか後でも読めます。折角保存したファイルが読めなくなる危険は避けられない面を持ちますが、自分の不注意でファイルを上書きして前の記録を壊す危険もあります。

テキストファイルとして扱うのが最も標準

ファイルの記録方式には種々あります。文字データをそのまま記録するテキストファイルで扱うのが最も基本です。テキストファイルの中身は、文字をコード化したものですが、英字用キーボードのキーに表示されている文字だけに限定したテキストファイルをアスキーファイルと言います。漢字やカナ文字は特殊なコード系を使いますので、アスキーファイルではありません。Plain_Basic のプログラムはアスキーファイルで扱うようにしてあります。パソコンメーカでは、自社のパソコン固有の記録方式を採用してユーザの囲い込みを図ることも行われましたが、結果的には姑息な対策であって、評判はよくありませんでした。アスキーファイルだけを扱うようにしますと、ユーザレベルでのファイルの扱いが単純になります。また、Plain_Basic 本体のソースコードの設計も楽になります。Plain_Basic では、グラフィックスにビットマップデータを扱うファイル処理を含めていません。グラフィックスは総て文字形式のステートメントで描くようにしてあります。この考え方のファイルをメタファイルと言います。Plain_Basic のグラフィックス処理プログラムは、一種のメタファイルと考えることができて、何回でも描き直しが効きます。

5. グラフィックスのプログラミング

5.1 グラフィックス技術の歴史概説

製図の自動化に始まったこと

科学技術の分野では、コンピュータを利用して図を描かせたいという要望は、数値計算を高速で処理したいと言う要望と表裏一体の関係にあります。数学では、グラフを描いて事象の性質を理解することは普通の方法です。科学的な測定には、線図で記録するレコーダが多く使われます。工業技術の分野では、大寸法の製図に応用する**自動製図**が研究されました。中でも、地図の作成に応用するときには作図精度を上げることに大きな努力がされました。これが CAD のソフトウェアの源流です。CAD は、二つの意義があります。一つは Computer Aided Design であって**コンピュータ支援設計**と訳しています。もう一つが Computer Aided Drafting であって、これに自動製図を当てます。自動製図を実現するには、コンピュータの計算結果を線図で描かせる大型作図装置の開発と、それを制御するプログラミングが必要でした。これは作図装置に依存しますので、今でいう**オブジェクト指向プログラミング**です。自動製図は、原理的にはペンを移動させて線図で図形を描かせます。ペンに代えて工作機械の工具を使うことは CAD と同質の開発になりますので CAM(computer aided manufacturing)と言い、コンピュータ支援製作と当てます。この両方は相互に関係が深いので、CAD/CAM として一つの専門扱いをします。図を描かせることに限ると、濃淡図や、カラーを使う作図は、当初、装置の製作が難しかったことも一つの原因でしたので、テレビのようなグラフィックス装置を使って観察（モニタ）する使い方が主流でした。これが、コンピュータゲームに応用され、結果的にグラフィックス技術の発展に大きく寄与しました。印刷に使うプリンタは、作図装置とは関係が薄いと思われていましたが、ドットインパクトプリンタ・レーザプリンタ・インクジェットプリンタが開発されて、印刷装置と作図装置の区別が無くなりました。そして、グラフィックスの利用は、科学技術の利用だけでなく、一般の事務処理計算にも応用されるようになり、通称で business graph という分野に育ちました。Microsoft Excel に組み込まれているグラフウイザードがその例です。

図を描かせる基本技法は線図と濃淡図

コンピュータを利用して図を描かせることの総称が**コンピュータグラフィックス**です。手で描くときの作図の技法は、線を描いて輪郭を作り、その内側を墨や絵の具で濃淡を付けます。この方法のコンピュータ化は、ハードウェア・ソフトウェア共に種々の開発がされてきました。ソフトウェア的に言うと、線を引いて図を描くことを英語で line drawing（**線図**）、ある区画を決めて、その範囲を色で塗り潰すことを painting（**濃淡図**）と言い、この二つは別扱いです。ある領域を他の領域と区別するとき、線図では塗り潰しに代えてハッチングで作図します。濃淡図の方は、濃淡の度合いを定量的に制御できないと同質の図を再現できません。そのため、濃淡図は、小さな点（ドット）の集合密度と点の大小を変えて表します。写真を印刷物に使うとき、細かな点の集合図形に直すことを**網掛け**と言います。この作図法は、滑らかな線を描くのが苦手です。点の大小とその疎密をまとめてコントラストと言います。そうすると、小さな点の大小表示ができて、密度の変えられるグラフィックス装置でないと微妙な濃淡図を表すことができません。線図は、線の太さを変えますが、濃淡を変化させませんので、コントラストは白か黒（0か１か）の区別だけです。文字も図形ですが、太さに変化を付けた線図で表示します。つまり、コンピュータグラフィックスは作図装置に大きく依存します。

ソフトウェアの標準化の試みと実際

コンピュータグラフィックスは装置に依存しますので、装置が変っても、プログラムを書き換えることを最小限にして、同質の作図が得られる方法が必要です。このために標準化の努力がされてきました。作図装置本体を制御する命令の種類は多くありません。「装置の選択をする・座標を指定して・線を引く・点を打つ・線または点を描かせるペンの種類を選ぶ」の五種類が基本です。これらを組み合わせれば、殆どの作図ができます。これだけではユーザのプログラミングの負担が大きくなりますので、種々の応用ルーチンが提案され、ソフトウェアも販売されています。Windows の環境では、Windows API を間接的に利用するように制限され、これが一種の標準になっています。これらは、それなりに便利ですが、欲を出すと切りがありません。Plain_Basic では、グラフィックスの基本的な使い方を理解できるように、上の五種類を実現するようなコマンドに限定してあります。これだけでも、かなり興味のある作品を作ることができます。これは例題を参照して下さい。

5.2　座標系の知識

座標幾何学の素養が必要

図形の性質を扱う幾何学は数学の一分野とされていますが、歴史的には代数学とは別種の学問体系でした。幾何学に数学的な方法を応用するようになったのはデカルト(1596-1650)以降です。デカルトは座標系を体系づけましたので、数学で利用する直交座標系を**デカルト座標系**とも言います。そして座標系を使って幾何の問題を扱うのを**座標幾何学**とも**解析幾何学**とも言います。コンピュータを利用して作図するには、まず、大元になる基準座標系を決めます。これを**世界座標系**(world coordinates)と言います。突き詰めて考えると天動説・地動説にも関係する大きな話題です。実践的には、自分の居る狭い範囲で、適当に世界座標系を決めます。世界座標系は三次元空間を考えるのですが、グラフィックスでは垂直な壁面に二次元の世界座標系を決めて、そこに在る図形を擬似的なカメラを使って写し取る、とするモデルを考えます。カメラのファインダはこの平面世界座標の或る矩形範囲の図形を切り取ると考え、その範囲を**ウインドウ**と呼びます。コンピュータ用語ではウインドウの用語がやたらと出てきて、グラフィックスの方のウインドウと混乱するようになりました。そのため、コンピュータのモニタ上に表示する枠の方は**フォーム**と言い換えています。さて、この写し取った写真のネガから、それをグラフィックス装置、ここではモニタ画面に、引き伸ばしなどの処理をして焼き付けると考えます。この画面全体には**装置座標系**(device coordinates)が決められています。図は、この全体画面の一部に枠を決め、そこに貼り込むようにします。この枠を**ビューポート**(viewpoit)と言います。そして、ここに、先のウインドウの座標枠が納まるように座標系を当てはめる**座標変換**(coordinate transformation)を行います。これを、**ウインドウ-ビューポート変換**と言います。作図の命令は、もとの世界座標系で測った図形の座標を使います。なお、先のフォームは、グラフィックス用語のビューポートに当たります。

数学座標系と装置座標系の考え方が異なること

手で図を描くとき、座標の考え方を必要としませんが、相対的な位置関係や大きさを表すために長さの基準は必要です。物差しとコンパスは基本的な作図の道具であって、これを使って描く図を用器画と言います。コンピュータグラフィックスは、一種の用器画です。この場合には、図の位置と大きさを決める基準が必要ですので、ここに座標系の考え方を使います。しかし、現実問題に応用する具体的な座標系の約束は、単純ではありません。コンピュータグラフィックスの画面はモニタを使うのが標準です。それはほぼ垂直な面です。座標の原点は左上にあり、右向き(X)と下向き(Y)に正の整数で座標を測ります。この尺度はピクセル単位ですので、実寸法ではありません。水平面に置いた用紙上に図を描くときは数学座標系（デカルト座標系）を考えます。その向きは右向き(X)と、手前から向こう向き(Y)を正の向きとし、マイナスの座標値も使います。単位も便宜的に考えます。空間的に考えると、X 軸 Y 軸の向きの定義が**左手系**と**右手系**との違いがありますし、Y 軸の向きが垂直と水平と違っています。そうすると、空間的に３番目の座標軸 Z を考えるとき、その向きも変ります。モニタ上の図形を印刷するときには、図の実寸法を正しく表す座標系と対応させなければなりません。

世界座標系を決めることから始める

作図は、描きたい図形の位置と寸法とを世界座標で考えることから始めます。図形が表現されている平面を仮想し、そこの世界座標系(X,Y)を数学座標系で考えます。Y 座標は上向きを正と約束します。この中に、図形が納まる座標範囲を指定します。これをウインドウの定義と言います。これには種々の方法が提案されていますが、Plain_Basic では DPWIND のコマンドで指定します。このモデルは、仮想のカメラを考え、そのファインダを覗いて、レンズの光軸を被写体の中央座標に合わせ、ズーミングまたはカメラと被写体との距離を調整して、ある範囲の矩形領域が収まるようにします。仮想カメラの光軸が狙う世界座標 Wx,Wy と、視野の横幅 Ww の三つの数値を世界座標で定義します。擬似カメラのフイルム寸法の**縦横比**（アスペクト比：aspect ratio)は、デフォルトとして横長の３：４としました。これは、モニタまたはテレビ画面一杯に図を貼り付けることを考えるからです。デフォルトのウインドウ枠は、原点を中央に置き、横幅を 640（-320<X<320)にしました。これらの値は、グラフィックスモニタの最小解像度が 480×640 ピクセルであることを含みにしたものです。

5.3　作図装置側の領域

ビューポートが実質的な作画領域である

作図領域を最大に使うときはモニタ画面全体です。実際は、Windows の枠、つまりフォームの枠の中に、タイトルバーやメニューバーの領域を除いた**クライアント領域**を当てます。また、この中に、さらに作図専用の枠を決めることもあります。ここがグラフィックスの用語で言うビューポートです。ビューポート内にはその左上を原点とした専用のピクセル座標を使って作図します。そのアスペクト比は擬似カメラで定義したフイルムのアスペクト比とは、同じではありません。そこで、ビューポートにフイルムから図形を焼き付ける方法が二通りあります。フイルムの縦横で引き伸ばし倍率を変えて、ビューポート全体を使う方法(an-isotropic projection)、もう一つはフイルムの縦横比を変えないで、ビューポート内で最大に貼り付けます(isotropic projection)。後者の方が標準です。そうすると、ビューポートの天地または左右に余分な領域ができます。フイルム図形がピクセルデータであると、そこは余白になりますが、線図はその領域にもフイルムが延長したように描くことができます。コマンド DPWIND は、ウインドウ-ビューポート変換の準備をします。

基本作図命令は線を引くコマンドであること

Plain_Basic のビューポートは、子ウインドウであるグラフィックスウインドウであって、そこにグラフィック専用の Canvas を一杯に載せてあります。元の座標系は左上を原点としたピクセル座標です。ここに図を描かせる命令は、プログラム開発に使った C++Builder から間接的に Windows API のライブラリを利用します。ユーザレベルでは、フォームの方の座標を気にしないで、世界座標を使った作図コマンドを使います。その基本コマンドは、**DPMOVE** と **DPDRW** の二つです。引き数は、それぞれ、線を描く始点と終点の座標 X,Y であって、世界座標の方で指定します。

幾つかの補助的なコマンドを使うこと

色の指定を除けば、線を描くコマンドがあれば、殆どの図形を作図することができます。使い易さを目的として、定型的なプロパティや単純な図形は、独立したコマンドにまとめます。「線の太さを変える・線種（実線・破線など）を変える・円を描く、点や記号を打つ・文字を書く」などです。欲を出すと切りがありません。Plain_Basic はプログラミングの入門の意義がありますので、最小限のコマンドしか準備しませんでした（表 5.1 参照）。これらの元になる参考資料は、主として Fortran や NEC の N88BASIC などで作成したサブルーチンです。サブルーチンの標準形は、例えば DPDRAW(X,Y)のように括弧を付けて引き数をコンマで並べます。Plain_Basic では、この括弧表現は関数の場合に使う仕様にしていますので、括弧を外したコマンド表現にしています。

表 5.1　標準的なグラフィックスコマンド

コマンド名	引き数	説　　明
DPERAS	なし	グラフィックス画面を消去する
DPWIND	WX,WY,WW	対象画面をカメラ光軸が狙う中心座標と視野の横幅。 デフォルトは、DPWIND 0,0,640 としてある。 縦横比は、デフォルトで 3:4 である。 この設定はメニューでも変更できる。
DPMOVE	X,Y	線を引かないでペン位置を設定する。 何も指定しなければ、以前に描いた図形の終端。
DPDRAW	X,Y	標準は、DPMOVE で指定した位置から線を引く。 線種、色などは、その前に指定した値が使われる。 デフォルトは、黒の細い実線。
DPCIRC	X,Y,R	中心座標 X,Y、半径 R の円を描く。塗り潰しはしない。
DPMARK	X,Y,IP	指定した座標に、IP で決めた番号の記号を描く。
DPENTX	IS	実線・破線・点線の線種を決める。
DPENSZ	IZ	線の太さをピクセル単位で決める。
DPTEXT	X,Y,"text"	最初の文字の左下の座標を X,Y として文字列を描く。

5.4　円の作図の例題

ダイレクトモードでも作図できること

Plain_Basic はインタラクティブに使うことができますので、キーボードから下のように入力するとすぐに円を描きます。この作図例は、第２章の始め、図 2.1 に示しました。

```
DPCIRC 0, 0, 100
```

毎回キーボードからテキストを入力しているのは面倒ですので、簡単なプログラミングをします。それは、ラベルを付けて、下のように入力します。

```
10 DPCIRC X, Y, R
```

キーボードから、X, Y, R に直接値をセットし、GOTO 文で実行させます。例えば、

```
X=-20
Y= 100
R= 200
GOTO 10
```

最後の GOTO 10 に変えて RUN とすると図は得られません。それは、先に入力した変数が総てクリアされるからです。

プログラムにまとめること

データを変えて、何度も同じ計算ができるようにすることがプログラミングです。上の例題を次のように入力してプログラミングします。

```
10 DPERAS
20 X=-20
30 Y= 100
40 R= 200
50 DPCIRC X, Y, R
```

このプログラムは、RUN と入力して実行すると、前の作図を消去して円を描きます。キーボードから、下のように一行にまとめた文を入力すると、別の円を描きます。

```
X=10: Y=-100: R=150: GOTO 50
```

なお、幾つかの例題は、付録Ｃの例題集を見て下さい。

付録Ａ：

教育利用を目的とした簡易なBasicのインタプリタ

Plain_Basic 言語仕様書

2006-08-12版

目次

予約語は付録Ｂ参照

A0.　はじめに

Plain_Basic とは、1980 年代の８ビットマイコンに搭載されていた簡易な Basic インタプリタの機能を Windows のパソコン上で再現するように作成したプログラムであって、筆者の命名です。この言語仕様は、８ビットマイコンを対象とした JIS 基本 Basic(JIS C6207-1982、現在は廃止)にほぼ準拠しています。Plain_Basic が Windows の環境で立ち上がると、図 A1 のようなウインドウ画面が現れ、これが、いわば独立したコンピュータシステムのような環境を提供します。画面は、上からタイトルバー、メニューバー、キーボードからの入力を受け付ける１行分のテキスト入力枠、その下に、DOS のモニタ画面を擬似的に表す二つの子ウインドウ（コンソールとキャンバス）が並びます。DOS の環境では、キャラクタディスプレイだけを使いますが、マイコンではグラフィックスディスプレイも同じモニタを使い、これをソフトウェアで切り替えて使用しました。Plain_Basic は、これらを、最初から専用の二つの子ウインドウに分けた MDI インタフェースを採用しました。この全体システムの中で、「OS 関連のコマンド、Basic プログラムのエディタ、ファイル処理、Basic プログラムの実行」など、コンピュータを扱う基礎的な処理ができるように設計しました。

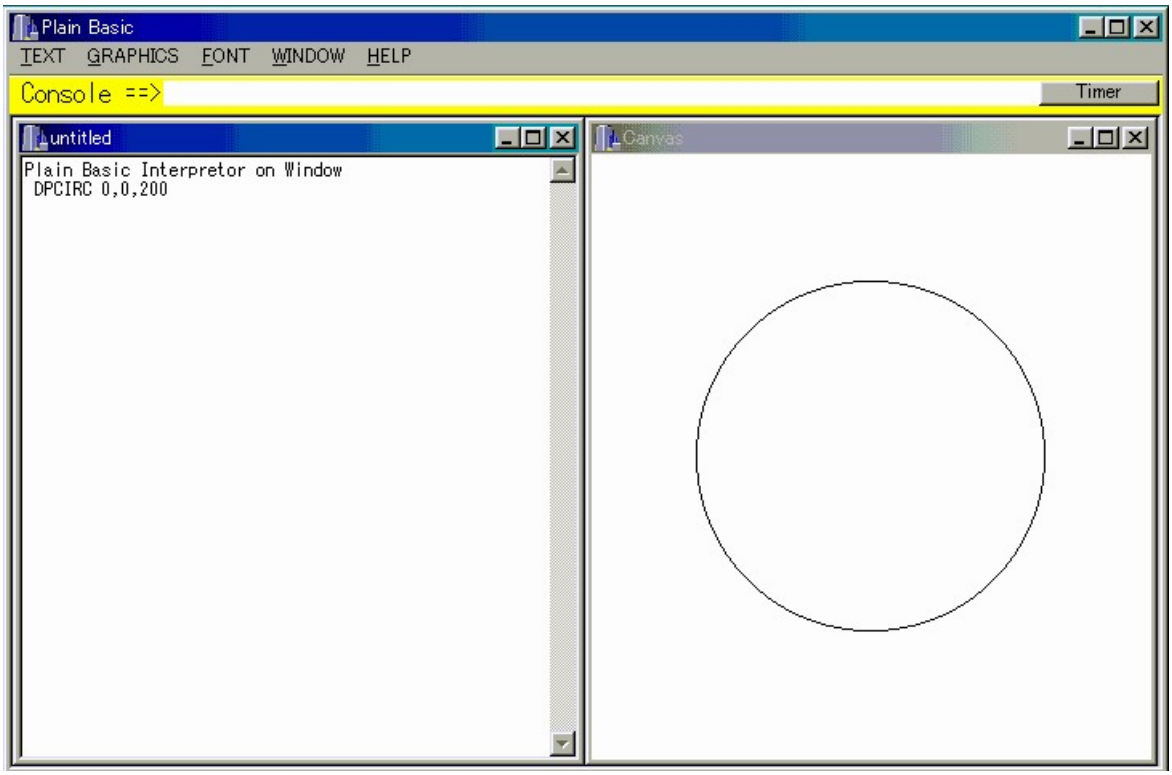

図 A1: Plain_Basic の立上げ時画面（円を一つ描かせました）

そもそも Basic とは、初心者のプログラミング教育を目的として開発された言語です。便利に使えると欲も出て来ますので、次第に高度な仕様が追加され、Visual Basic に見るような高級言語に変身し、初心者には非常にハードルの高いツールになってしまいました。したがって、本来の、初心者教育に向いた Basic 言語を意識したのが Plain_Basic 開発の一つの目的です。とは言っても、曲がりなりにもプログラミング言語です。一寸した数値計算には、関数電卓よりも使い易いプログラミングツールになりますので、科学技術計算にも充分の利用ができます。Plain_Basic 単独は、言語仕様を単純にして基本機能だけに制限したバージョンです。これをプロトタイプとして、科学技術計算に使うサブルーチンなどを組み込んだ拡張 Basic 版が作成できます。この Plain_Basic は、教育利用を兼ねていますので、簡単なグラフィックスルーチンを組み込んだバージョンです。幾何モデリングのライブラリを組み込んだバージョンには GEOMAP+Basic などがあります。これらのマニュアルは、Plain_Basic に関係したコマンドの説明を省いてありますので、この言語仕様を合わせて参考にして下さい。

A1. Plain_Basic 操作の概要

A1.1 Plain_Basic は、小単位の**コマンド駆動型プログラム**(command driven program)の集合です。このユーザインタフェースは古典的な CUI 方式で設計されています。Windows の環境ですので、メニューを使う処理もありますが、それは Plain_Basic の主な利用とは異なるオプション的な作業に当ててあります。

A1.2 Console のラベルが付いたテキスト入力窓(**Edit**)をアクティブにすると、Plain_Basic は、キーボードからの文字入力を受け付けます。一行単位の文字並びでコマンドやステートメントを入力すると、直ぐに実行します。この状態を**ダイレクトモード（直接モード）**と言います。コンソール画面は、単純なテキストエディタの機能を持つ **RichTextEdit** が載せてあって、キーボードからの入力テキストを**エコー**表示します。

A1.3 毎回、キーボードから文字列を入力する手間を省くため、文字列を**内部プログラム**に保存しておいて、まとめて連続実行(**RUN**)させることができます。内部プログラムの作成は、ステートメントの頭に整数の行番号（**ラベル**）を付けてキーボードから入力します。

A1.4 歴史的な理由があって、コマンドやステートメントに使う英字は大文字で表現してあります。ユーザレベルでの文字入力は、大文字・小文字を区別しません。

A1.5 内部プログラム文は Basic 文の文法で作成します。実行順序に合わせて昇順に、1-32767 の範囲で整数の行番号ラベルを付けます。番号付け作業を助けるために、**AUTO** コマンドを入力すると、行番号を昇順に自動的に発行しますので、本文のステートメントだけを入力します。この状態が（**プログラム）編集モード**です。編集モードから抜け出すには **EXIT** と入力します。

A1.6 作成したプログラムの確認は **LIST** コマンドでテキストウインドウに書き出して行います。このプログラムをテキストファイルに保存するには **SAVE** コマンドを使います。逆に、テキストファイルからプログラムを読み込むのは **LOAD** または **MERGE** コマンドです。

A1.7 ダイレクトモードで **RUN** コマンドを入力すると、変数などをクリアして初期状態にして（**プログラム）実行モード**に入ります。内部に保存されたステートメントを先頭から順々に読んで、これを解読して実行します。**GOTO** 文は、指定された行番号のステートメントから実行させるコマンドとして使用できます。このときは、変数などのクリアは行いません。

A1.8 一部のステートメントは、複数のステートメントの組で実行しなければならないものがあります。例えば、**FOR-NEXT, GOSUB-RETURN** などです。これらのステートメントは、ダイレクトモードでは実行できません。また、**AUTO, EXIT, LIST, LOAD, SAVE, RUN** などはコマンドですので、逆にこれらをプログラム文に書いて実行させることはできません。

A1.9 コマンドを含め、ユーザがキーボードから入力したそのままを記録したテキストファイルを作って、それを入力させて実行させることを**バッチ処理**と言います。Plain_Basic では、バッチファイルからテキストを読み込むコマンドは **DECK** です。用語としては一寸古いのですが、カードや磁気テープなど、外部に作られたデータをこのように呼ぶ習慣があったためです。この機能があると、ユーザがキーボードから解放されて、デモンストレーションを行うことができますので、教育用にも利用できます。

A1.10 バッチ処理を行わせると、何が現在進行中であるかが分からないことがあります。これを助けるため、**エコー**をコンソールウインドウに表示するようにできます。エコーという用語は、端末機のキーボードから通信回線を使ってデータをホストコンピュータに送ったとき、実際に正しく受信されたことの確認をとるため、同じデータを送り返して表示する機能を言いました。**ECHON/ECHOFF** のコマンドは DECK ファイルから入力されたテキストを、コンソールウインドウに表示するか、しないかを切り替えます。

A1.11 実行モードのとき、何行目のプログラムが実行されているかをモニタする機能を**トレース**と言います。**TRON/TROFF** のコマンドはトレースの機能を「使う・使わない」を切り替えます。トレースは、実行しているプログラム文の整数ラベルをテキストウインドウに表示します。エコーとトレースは、プログラム文の実行状態をモニタしますので、プログラム文の**デバッグ**に応用することができます。

A1.12 キーボードからの文字入力については、システムに依存する制限が種々あるため、非常に基本的な条件で設定しています。一行に書ける文字数は、半角約 127 文字以内(80 文字以内が手頃です)、変数や配列名は 15 文字までの英数字の並びまで、文字列変数も 15 文字までの固定長です。因みに、最近のプログラミング言語では長い変数名が使えるようになりましたので、一行に許される文字数が長くなっています。

A1.13 文字列のカット・コピー・貼り付けなど(Cut,Copy,Paste)の基本的な処理は Windows の Edit メニューの定番ですが、これは Ctrl キーを併用するキーショートカットが使えますので、意図的にメニューには含ませてありません。

A1.14 Plain_Basic のプログラム文は 1 行で完結させ、次の行に論理的に継続するように分割できません。しかし、短いプログラム文を、コロン（:）を挟んで書くマルチステートメントができます。これは、プログラムリストの行数を節約し、見易さを助けることができます。

A1.15 Plain_Basic は長大なプログラム文を作成して実行することを考えていませんので、プログラム文全体の文字数は最大で 30KB 程度の制限があります。これより大きくなるプログラムは、MERGE コマンドを使うことで対処できます。

A1.16 Plain_Basic の言語仕様は、FORTRAN の言語仕様を真似た部分が幾つかあります。まず、暗黙のデータ型として、型宣言なしで英字の I-N で始まる変数名や配列名は整数型、それ以外は実数型です。配列のデータ並びは、列の並びです。配列の要素を示す記号は、丸括弧()ではなく、C/C++言語に倣ってブラケット[]にしました。例えば配列 A[2,3]は、メモリの中では A[1,1], A[2,1], A[1,2], A[2,2], A[1,3], A[2,3]の順で並びます。また、配列の基底は 1 です。

A1.17 Plain_Basic では、配列名を宣言なしに使うこともできます。このときのデフォルトの配列寸法は 3 に設定されています。

A1.18 8 ビットマイコン時代の Basic は、長い変数名が使えなかったので、短い英字名に記号の添字を付けて型を区別する方法が使われました、例えば、A%, A!, A#, A$は、それぞれ整数型、実数型、倍精度実数型、文字列型の変数名です。この仕様のうち、Plain_Basic では文字型を表す$だけを採用しています。添字記号を付けた全体の変数名も 15 文字までです。

A1.19 キーボードから直接入力できる文字種は、それ程多くはありません。歴史的には、キーボードはテレタイプライタ(TeleTYpewriter)で利用できる文字・数字・記号に限られていました。これを **TTY コード**と言い、英字の小文字を使わないコード系です。Plain_Basic は英小文字も使えます。漢字などの日本語は、ローマ字で入力し、カナ漢字変換をしますので、直接に入力できる文字ではありません。しかし、二つの引用符""で囲めば、**PRINT** 文で利用できます。

A1.20 コマンド入力型のプログラムは、コマンド名を忘れると何もできません。この不便を解消するため、**HELP** の機能をメニューから使えるようにしました。

A1.21 Plain_Basic を使い慣れてくると、便利な機能が欲しくなります。それは、ユーザにプログラミングの知識が増えたことですので、教育上で効果があったと考えることができます。このことを配慮して、Plain_Basic では、敢えて古典的で保守的な機能に限定してあります。便利な機能への要求は専門分野と関係しますので、Plain_Basic から特化した幾つかの発展バージョンがあります。幾何の計算を便利にした Geometry_Basic(G_Basic)、幾何モデリングのサブルーチンパッケージを組み込んだ GEOMAP+Basic などがそうです。これらのユーザーズマニュアルでは、この Plain_Basic のマニュアルを基礎参考資料としています。

A2. 文字セット

A2.0 Plain_Basic の言語仕様は、文字セットに ASCII コードを使います。初期の８ビットマイコン時代の Basic インタプリタは TTY コードを使いましたので、文字種はかなり限定されていました。Plain_Basic では、その制限を少し緩やかにしました。特に、配列の表現に、丸カッコ()に代えて角カッコ[]を使うようにしたのが大きな変更の一つです。

A2.1 英字　　Ａ－Ｚ、ａ－ｚ，（半角の英字です。大文字・小文字の区別をしません）

A2.2 数字　　０－９

A2.3 特殊文字　　（記号）は表 A2 に示します

A2.4 二つの引用符 〝〝 で囲んだ文字列には、使用するシステムが許す文字を書くことができます。例えば、カナ、ひらがな、漢字などです。

A2.5 字形を持たない制御コード、例えば、BREAK, STOP, CTRL, ESC, END OF FILE,行末記号、改行記号などはシステムの定める入出力仕様によります。

表 A1　Plain_Basic で使用する記号

記 号	呼 び 方	用　途
	空白	文字、名前、数字などのの区切り
_	アンダースコア	英数字名の繋ぎに使い、英字扱いとします
,	コンマ	文字、名前、数字などのの区切り
.	点	数字の小数点
;	セミコロン	プリント文のリスト制御
:	コロン	マルチステートメントの区切り
＋	プラス	加算記号
－	マイナス	引き算記号、符号反転
＊	星印	積、掛算の記号
／	スラッシュ	数の割算記号、数字列の区切り記号
^	グラーブアクセント	べき乗記号
（	左括弧	関数の引き数表記、数式の優先順位を決めます
）	右括弧	（同上）
[	左角カッコ	配列の添え字を表すときに使います
]	右角カッコ	（同上）
“	引用符	文字列の引用（漢字は引用符で囲います）
$	ドル記号	文字型変数を示す添字記号
＝	等号	代入文、関係演算子
＞	より大きい	関係演算子
＜	より小さい	関係演算子
&	アンパサンド	論理積の記号
｜	縦棒	論理和の記号

表 A2　Plain_Basic で使用しない記号（システムが使う記号があります）

記号	呼 び 方	説　明
!	強調記号	
＃	番号記号	ファイル入出力を使いません。
%	パーセント記号	
‘	アポストロフィ	
～		
￥		アスキーコードは逆スラッシュで表記されます。
？		
{ }	波括弧	
@	アットマーク	URL などで用いるので使いません。

A3. 語(WORD)

A3.1　語とは英字で始まる 15 文字以内の英数字の並びです。英数字だけで構成する語は通称で名前と呼び、15 文字を越える名前はエラーとします。 $ 記号は文字型の変数名を表す接尾字として名前の構成文字に使うことができます。漢字は名前に使うことはできません。

A3.2　変数や配列には、分かり易い名前を付けますが、単語をハイフン(-)や点(.)で繋ぐことを許しませんので、その代わりにアンダースコア(_)を英字並に使うことができます。

A3.3　語の始まりと終わりの識別は、空白、コンマ、などの特殊文字です。

A3.4　Plain_Basic で用いる予約語（キーワード：Keyword）は、Basic 言語のプログラム文（ステートメント）に用いるキーワードと、オペレーティングシステム(OS)に属する制御命令（コマンド）を含めてあります（表 A４参照）。

A3.5　予約語以外の任意の英数字並びは、変数や配列の名前として用いることができます。

表 A3　Plain_Basic で用いる予約語

1	プログラム編集	AUTO, DELETE, EXIT, LIST, RENUM
2	プログラムファイル	LOAD, MERGE, SAVE
3	プログラム実行	CONT, END, NEW, PAUSE, QUIT, REM, RUN, STOP
4	バッチファイル	DECK
5	デバッグ機能	ECHON/ECHOFF, TRON/TROFF
6	型の定義	DEFINT, DEFDBL, DEFSTR
7	変数と配列の管理	DIM, ERASE
8	データ入出力文	PRINT, DATA, READ, RESTORE
9	プログラム実行制御	IF/THEN/ELSE, FOR/TO/NEXT, GOSUB/RETURN, GOTO
10	組み込み関数	ABS, ATN/ATN2, COS, EXP, LOG, MOD, RND, SGN, SIN, SQR, TAN
11	グラフィックス	CLS, DPCIRC, DPDRAW, DPENSZ, DPENTX, DPMARK, DPMOVE, DPTEXT, DPERAS, DPWIND

【備考】言語マニュアルは付録 B にまとめました。

A4. プログラムの行と文

A4.1　行は、キーボード、ファイル、などから転送される文字列のレコードであって、Plain_Basic で実行するコマンドまたはプログラム文の最小単位です。Plain_Basic のプログラムとは、この様な行の集まったものを言います。

A4.2　一行の物理的長さは、最大 127 バイトです。80 バイト以内が扱いやすい長さです。意味のある文字列を二行以上にまたがって書くことはできません。

A4.3　一行は、行番号、文（ステートメント）及びコロン（：）で作られ、二つの書式があります。

(1) 行番号　文　[：文　[：文　[：文 ……]]]

(2)　　文　[：文　[：文　[：文 ……]]]

A4.4　行番号はその行につけられたラベルを表し、正の整数で、1-32767 の範囲とします。

A4.5　文は、予約語で始まる文字列で Plain_Basic の実行単位です。代入文(assignment)は、変数名で始め、イコール記号(=)の右に式文(expression)が来る文です。算術式、関係式、論理式は文として扱い、式文です。

A4.6　書式 (1)で書かれた行は、そのラベルの番号の昇順に、内部記憶領域に記録されます。既に同じラベルで記録された行があれば、新しい行に書き替えられます。行番号だけが転送された場合は記憶領域にある同じラベルの行を消去します。

A4.7　書式(1)の文に用いることができない予約語があります。プログラム編集に用いる予約語、プログラムファイル用の予約語、プログラム実行用の予約語のうち CONT,NEW,QUIT,RUN、それと DECK がそうです。これらは、プログラム実行モードで現れとエラーになります。

A4.8　書式(2) で書かれた行は、ただちに実行されます。コマンドは、原則として、書式(2)で書きます。プログラム実行制御の予約語は書式(2)ではエラーとします。

A4.9　書式(1) で作成された行の集合は、**RUN, CONT, GOTO** コマンドによって実行されます。

A4.10　**AUTO** コマンドが実行されると、Plain_Basic は自動的に行番号を発生させることができますので、書式(2) が書式(1) を代行します。この状態が**プログラム編集モード**です。このモードの解除は、 **EXIT** を入力させます。

A5. 変数名とデータの型

A5.0　変数は、定義と宣言で確定します。定義は Plain_Basic 側で準備する約束です。Plain_Basic では複素数型、論理型の定義はありません。変数は宣言があって始めて実体を持ちます。高級プログラミング言語では定義と宣言の約束を厳格に理解してコーディングしなければなりませんが。Plain_Basic では、変数は、変数名が最初に現れたときに宣言が行われたと解釈して実体が作られます。高級言語を覚えるときには、この習慣の違いを理解しなければなりません。

A5.1　変数は、名前で引用します。Plain_Basic で使用する変数の型は、表 A5 に示す 3 種類です。

表 A4　変 数 と 型

番 号	型 の 名 前	寸 法	内　　容	初期値	添字記号	型定義文
1	**整数**	1	**8 バイト長**	0		DEFINT
2	**実数**	1	**8 バイト長**	0		DEFDBL
3	**文字列**	2	**15 バイト長**	**空白文字**	$	DEFSTR

備考：寸法とは、8 バイトを 1 ワードとして計算した記憶単位です。

A5.2　型定義文は、使用する変数名の頭文字で、その変数名をどの型にするかを決めます。

A5.3　型定義文は、それが宣言された時点で効力が始まります。先行する型定義文無しに用いる変数のデフォルト型は、I,J,K,L,M,N で始まる名前が整数型、それ以外は実数型です。

A5.4　英数字名の最後に付けた $記号は名前の一部とみなし、頭文字種に関係なく、文字型を持つ変数名になります。

A5.5　配列の宣言は **DIM** 文で行います。配列の下限値は 1 です。配列の次元数は 4 までです。単純な変数は、次元数 0 の配列と解釈されます。

A5.6　DIM 文の宣言なしに配列を引用できます。この時の配列上限値は 3 です。

A5.7　既に宣言された変数名、配列名を消去するには **ERASE** 文を使います。

A6. 変数への値の代入

A6.1　数は、0,1,-654、3.456, 1.2E-5、 0.234E3 の様に表します。 2 進数、 8 進数、16 進数の表現は使いません。

A6.2　数値データの入力、並びに計算は倍精度数（約 16 桁）の精度で行なわれます。

A6.3　PRINT 文での数値の出力精度は、デフォルトの書式変換を使い、約 10 桁です。

A6.4　文字データは、例えば"Basic123"のように引用符で囲み、最大半角 15 バイトです。

A6.5　変数、配列へ数値を代入するときは、代入文または READ 文を使います。ファイルからのデータ入力要請に使う INPUT 文は定義してありません。それに代わる方法として**内部ファイル**の概念を使い、これを読み出すのが READ 文です。これは次の様に使います。

READ　変数名並び　＜＝＝＝＞　DATA　数値の並び

A6.6　変数名並びとは、単純変数名、添字つき配列名、添字なし配列名を、コンマで区切って並べたリストです。添字なし配列名に要求される数値の個数は、その配列の寸法個数です。

A6.7　数値の並び（リスト）は、次の様に表すことができます。
(1) 1.2,1.3, 5　 6 7,10. 3.14 ；コンマでなくても空白で区切れます
(2) 3 TO 10, 1 TO 0 STEP -0.1 ；FOR-NEXT 文の仕様でくり返す指定です。個数の勘定を間違えない様にします。
(3) 10*6.0, 100*1, 3*0 ；　 n 個×同じ数値の意味です。
(4) 5*/,7,8；最初の 5 個分の領域を読みとばして次の二つに 7,8 と代入します。同じ変数並びに入力させるとき、部分的に以前の数値を生かすときに使います。
(5) 5,6,7/；３個の数値を代入して終とします
(6) ″test″；文字データは 15 文字以内を単位として、引用符で囲みます

A6.8　変数に代入される数値は、その変数の型に合わせた標準化が行われます。

A7. 式とその評価

A7.1　式(expression)とは演算子を介した変数、数値の並びを演算子で繋いだ文字並びを言います。添字なしの配列名、文字型を含めません。一つの意味を持つ式を括弧()で囲むことができます。

A7.2　Ｙ＝Ａの形を代入文と言います。Ｙは変数名、または添字付きの配列名です。記号＝は代入演算子、Ａが式です。記号＝は関係演算子としても使いますが、行と文の構造でインタプリタが区別します。

A7.3　式は、代入文の右辺、関数の引き数となる角括弧の中、配列の添字、条件文などに使います。

A7.4　式の評価は左から右に処理されます。演算子の優先順位は次の様に決めてあります。
(1) 括弧（　）の中の計算
(2) プラス（＋）とマイナス（－）を除くすべての演算子
(3) ＋および－

A7.5　演算子は、その左右にある変数の間、もしくは、その前に処理された結果との間に演算を行わせます。演算の種類と、演算子の記号を表 A6 に示します。

A7.6　演算子の両側にある二つの数値の型が異なるときは、整数→実数の順に自動的に型変換をして処理されます。組み込み関数の引き数は実数型です。

A7.7　ベキ乗優先順位は乗除と同じですので、ベキ乗を含む計算は、括弧を適当に使用します。例えば、下に例を示します。
2*3^2=36、　 2*(3^2)=18,　 2^3*2=16,　 2^(3*2)=64

A7.8　条件式は、関係演算子を使って、例えば A>B のように書きます。この条件が成立すれば、つまり真であれば、結果は整数の-1 が求まります。もし不成立であれば偽であると言い、結果は整数の 0 が求まります。

A7.9　文字を使う演算子は使いません。例えば、AND,OR,XOR,NOT は定義してありません。

表 A5　演算子の記号

区 分	記 号	用　途
算術演算子	＋	足し算
	－	引き算、マイナス１を掛けて足し算することと同じ。
	＊	掛け算
	／	割り算
	^	べき乗
関係演算子	＝	等しい･
	＞	より大きい
	＜	より小さい
	＞＝	等しいか、もしくは大きい（以上）
	＜＝	等しいか、もしくは小さい（以下）
	＜＞	等しくない
論理演算子	＆	論理積
	｜	論理和

A8. 組み込み関数

A8.1　Plain_Basic にあらかじめ組みこまれた関数の名前を表 A4 に示してあります。
A8.2　関数の引き数は、実数型に変換されて計算に用いられ、関数の結果も実数型で返されます。
A8.3　文字および文字列を扱う関数は用意してありません。

A9. Plain_Basic の操作方法

A9.1　制御コマンドとは、オペレーティングシステム（OS）に依存して仕様に制限を伴うコマンドを言います。特に、プログラムの開始、終了、割り込み、ファイル処理、エラーによる中断とその回復などに関係します（表 A4 の１～４が特に関係があります）。
A9.2　Plain_Basic の実行時は、四つの状態（モード）があります。
　(1) 直接モード（ダイレクトモード）　：（コマンドの入力を待っている）
　(2) プログラム編集モード　：（自動的に行番号を発行します）
　(3) プログラム実行モード　：（内部のプログラムを実行している）
　(4) バッチ処理モード　：（バッチファイルを読んで実行している）
A9.3　直接モードは最も基本的な状態です。Plain_Basic は、ターミナルにプロンプトを出力し、ユーザからのキーボード入力を待ちます。第２節文字セットに示した以外の文字が使われた場合は、システムの定義に従います。他のモードになる時は、必ず一度直接モードに戻ります。
A9.4　Plain_Basic の実行の終了は、親ウインドウの閉じるボタンで行うことができますが、**QUIT** コマンドで終了します。
A9.5　**AUTO** は、直接モードから編集モードに変えるコマンドです。システムは、自動的に行番号を発行し、プログラムステートメンの入力を待ちます。番号の出力とプロンプトのデザインはシステムの仕様に影響を受けます。
A9.6　編集モードから、直接モードへ帰るためのコマンドは **EXIT** です。Ctrl キーを使ったキーショートカットは使用しません。
A9.7　プログラム実行モードへ制御を渡すコマンドは、次の三つです。
　(1)　**RUN**　：データを初期化して実行
　(2)　**CONT**　：STOP コマンドで中止した次の行から再開
　(3)　**GOTO** 行番号　：データの初期化をしないで指定した行から実行

A9.8　プログラム実行モードから直接モードへの復帰は、次のときです。
(1) プログラムの実行が　END 文のある行にきたとき
(2) プログラムの実行が STOP 文のある行にきたとき
(3) プログラムの実行が最終行を越えたとき
(4) Plain_Basic で判断したエラーが生じたとき

A9.9　プログラムが無限ループに入って出られないときなど、途中でプログラム実行モードを出たいとき、システムの強制終了以外 Plain_Basic には適切な方法がありません。

A9.10　バッチ処理モードとは、ユーザが直接キーボードからデータを入力させることに代えて、あらかじめファイルに作成したデータを読み込んで実行させるモードを云います。デモンストレーションや、教育に応用する際、人が直接操作する必要がありません。バッチ処理は、直接モードで、DECK　″ファイル名″と入力します。

A9.11　Plain_Basic は、ホストコンピュータと通信回線で結んだリモート・グラフィックス・ターミナルで利用すること考慮していました。ECHON は、バッチファイルからのデータ入力の状態を監視するため、エコーを取り、ターミナルにデータのリストを表示します。ユーザが直接キーボードからデータを送ると、同じデータがホストコンピュータから送られてきますから、送信データのテストができます。エコーの機能を止めるには ECHOFF とします。

A9.12　プログラム実行モードでは、GOTO/GOSUB などで制御が別の行に移らない限り、行番号の順に実行します。どの行を実行中であるかを知ることをトレースと云います。実行時にこの行番号のリストをさせるコマンドが TRON 、中止が TROFF です。

A9.13　Plain_Basic は、コマンド駆動型のプログラムですので、コマンド名を忘れると何もできません。そのため HELP の機能は必須です。HELP はメニューで参照します。

A9.14　ヘルプ用のファイルには日本語用と英語用とがありますので、ユーザの希望に合わせて、選ぶことができます。

A9.15　Plain_Basic の実行時にユーザの処理エラーがあるとき、エラーメッセージを出してダイレクトモードに戻ります。エラーメッセージは差し換えができません。現在は英語のメッセージにしてあります。

A10. ファイルの制御

A10.1　Plain_Basic の入出力のハードウェア最小構成は、キーボード、ディスプレイ、およびディスク装置の三つです。

A10.2　能率的な使い方をするとき、下のようなテキストファイルを利用します。
(1)　LOAD/SAVE/MERGE コマンドで利用するプログラム保存のファイル
(2)　DECK コマンドで呼び出すバッチ処理用のテキストファイル

A10.3　ファイルの入出力は、使用するシステムのＯＳに依存しますので、Plain_Basic から OS のコマンドをどの程度利用できるかが、ファイルの使い勝手の良さを決定します。DOS の環境だけで利用するシステムは、ディスクファイルのディレクトリを実行中に読むことが出来ませんでした。そのため、ファイル名を忘れてしまうと手の施しようがありませんでした。Windows の環境では、エクスプローラを並列に走らせることができますのでファイル名の確認が簡単にできるようになりました。LOAD/SAVE/MERGE でファイル名を省略するとダイアログボックスを表示してユーザの入力を待つようにしてあります。

A10.4　ファイルのアクセスに使うファイル名を直接利用するときは、二つの引用符""で囲みます。拡張子の種類を区別しませんので、ファイル名と拡張子とをすべて書きます。パス名などの付け方はシステムの仕様に従います。ファイル名全体の長さは 40 字（バイト）以内の制限があります。

A10.5　内部プログラムの保存と読み出し(LOAD/SAVE/MERGE)に使うテキストファイルでは、習慣として拡張子.BAS を使います。しかし、Windows の環境では、この拡張子は Visual Basic のファイルと区別できなくなりますので、.txt が無難です。　（この章終わり）

付録 B：

教育利用を目的とした簡易な Basic のインタプリタ

Plain_Basic キーワード及びメニュー マニュアル

2006-08-12 版

目次

アフファベット順のキーワード目録

キーワード	分類	キーワード	分類	キーワード	分類	キーワード	分類	キーワード	分類
ABS	B10	ATN	B10	ATN2	B10	AUTO	B1	CLS	B11
CONT	B3	COS	B10	DATA	B8	DECK	B4	DEFDBL	B6
DEFINT	B6	DEFSTR	B6	DELETE	B1	DIM	B7	DPCIRC	B11
DPDRAW	B11	DPENSZ	B11	DPENTX	B11	DPERAS	B11	DPMARK	B11
DPMOVE	B11	DPTEXT	B11	DPWIND	B11	ECHOFF	B5	ECHON	B5
ELSE	B9	END	B3	ERASE	B7	EXIT	B1	EXP	B10
FOR	B9	GOSUB	B9	GOTO	B9	IF	B9	LIST	B1
LOAD	B2	LOG	B10	MERGE	B2	MOD	B10	NEW	B3
NEXT	B9	OPTION	B12	PAUSE	B3	PRINT	B8	QUIT	B3
READ	B8	REM	B3	RENUM	B1	RESTORE	B8	RETURN	B9
RND	B10	RUN	B3	SAVE	B2	SGN	B10	SIN	B10
STEP(FOR)	B9	STEP(DATA)	B8	SQR	B10	STOP	B3	TAN	B10
THEN	B9	TO(DATA)	B8	TO(FOR)	B9	TROFF	B5	TRON	

B1. プログラム編集

解説

Plain_Basic のプログラム文は、直接モードで、行頭にラベル（行番号）を付けてコンソール入力枠から入力します。プログラム文は、番号の昇順で内部メモリに書き込まれます。行番号を毎回入力する手間を省くため、行番号の自動発行のモード（プログラム編集モード）があります。コマンドの AUTO を入力すると、プロンプトの表示が変って、整数の番号が発行されますので、入力枠に数字番号無しでプログラム文を入力します。プログラム編集モードから抜けるときは、コマンドの EXIT を入力すると直接モードに戻ります。テキストファイルでディスクに保存(SAVE)、読み出し(LOAD/MERGE)ができます。ファイルの拡張子は、.bas を良く使いますが、Visual Basic のソースコードと衝突しますので、ユーザの方で適当に決めることができます。.txt が無難でしょう。読み出しのとき、フォルダ名やファイル名が間違っているとき、または、ファイル名を省略しても、コモンダイアログボックスを表示しますので、そこでファイル名の指定ができます。ファイル名の英字は、大文字・小文字を区別しません。

プログラム文の全部または部分を見たいときに、コマンドの LIST でテキストウインドウに出力できます。プログラム文の修正や追加は、行番号を付けて入力します。番号だけを入力すると、その番号ラベルの文を消去します。部分的または全部のプログラム文を消去するときに DELETE を使います。

行番号を付け直すときに RENUM を使います。ただし、プログラム文中の GOTO 文、GOSUB 文のラベルは、現在のバージョンでは変りませんので、注意して修正しなければなりません。修正作業などは、テキストウインドウの文字並びをコピーして、入力枠に貼り付ける編集機能を利用することができます。

AUTO [starting_line][,increment]

編集モードにして、自動的に行番号を発生します。省略値は AUTO 10, 10 と同じです。

例： AUTO 100, (100 から始めて 10 刻みの番号を発行します。)

例： AUTO , (初期値はデフォルト値の 10 で、30,50,...と発行します。)

例： (初期値はデフォルトの 10 で、20,30,40...と発行します。)

EXIT

編集モードから直接モードに戻ります。

DELETE [starting_line][- endline

プログラム文の指定された番号の行を削除します。指定を省略すると全ての行が削除されます。

例: DELETE　　(行番号 100 の一行だけを削除します)

例: DELETE 100-800　(行番号 100 から 800 の範囲にある行を削除します)

RENUM [new_line_number] [[,old] [,step]]

古いプログラムの行番号を付け換えます。番号指定を省略すると、AUTO 10,10 と同じ並びになります。途中から若い番号に付け替えても、全体としての並びは変りません。そのときは、一旦仮の名前で SAVE し、改めて LOAD し直すと昇順に揃います。

例: RENUM 10, 100, 10　　(100 番の場所を 10 とし、以降を 20,30,40 とします)

LIST [starting_line] [- endline]

Plain_Basic のプログラム文をテキストウインドウに書き出します。番号指定を省略すると全プログラム文をリストします。

例: LIST 70-90

例: LIST 120

B2. プログラムファイル

解説

Plain_Basic のプログラムは、テキストファイルでディスクに保存(SAVE)、読み出し(LOAD/MERGE)ができます。ファイルの拡張子は、.bas を良く使いますが、Visual Basic のソースコードと衝突しますので、ユーザの方で適当に決めることができます。.txt が無難でしょう。読み出しのとき、フォルダ名やファイル名が間違っているとき、または、ファイル名を省略しても、コモンダイアログボックスを表示しますので、そこでファイル名の指定ができます。ファイル名の英字は、大文字・小文字を区別しません。

LOAD "program_file_name"

ディスクに保存されている Plain_Basic プログラムを読み込みます。前にメモリにあった Plain_Basic プログラムは消去されます。この時点では、前に実行したプログラムの変数や配列はそのまま残っています。

例:　LOAD "PROG12"

例:　LOAD "B:TEST.BAS"

SAVE "program_file_name"

メモリ上にある Plain_Basic プログラムをディスクに保存します。

例: SAVE "PROG12"

MERGE "program_file_name"

ファイルから Plain_Basic プログロプログラムをロードし、現在メモリにある BASIC プログラム文に追加します。同じラベルの行があれば置き換えます。Plain_Basic の実行文だけをまとめたファイルと、DATA 文だけを別ファイルにしておいて MERGE するような使い方ができます。DATA 文だけのファイルを差し換えると、データファイルの読み込みと同じ処理ができます。

例: MERGE "PROG23"

B3. プログラム実行

解説

Plain_Basic のプログラムは RUN コマンドでプログラム実行モードに入り、内部プログラムを読んで順に実行します。プログラム行の終に来るか、END コマンドの行に来ると、インタラクティブモードに戻ります。プログラムの実行を一時的に停止させる方法は二つあります。STOP 文と PAUSE 文です。プログラム文の中に、何もしない、読み飛ばしを指定する REM 文があります。このコマンドの後に任意の文字を書いて、コメントとして利用することができます。Plain_Basic 本体の終了は、QUIT コマンドで行います。また、 メインフレームの閉じるぼたん、左上のシステムアイコンをクリックしても終了します。

NEW

現在メモリにある BASIC プログラムを消去し初期状態にします。グラフィックスの設定値も初期状態にします。

RUN

変数、配列などをすべて消去して初期状態にしてから、現在メモリにある Plain_Basic プログラムを最初の行から実行します。

STOP

プログラムの実行を停止し、その場所の行番号ｎを ″BREAK IN ｎ″ とコンソールに表示します。再開は CONT と入力します。

CONT

プログラム中の STOP 文によって停止したプログラムの実行を、STOP 文の次から再開します。MERGE コマンドや、プログラム文の変更をしたときには、再開できない(can't continue)のエラーメッセージがでます。

END

実行中の Plain_Basic プログラムがこのコマンドに来ると終了し、ダイレクトモードに戻ります。プログラム行の最後で終了するのであれば、END 文は必要ありません。

REM any comments list

このコマンド以降行末までは読み飛ばしますので、任意のコメント(remarks)を書き込んでおくことができます。

PAUSE [itime]

itime 秒だけ時間待ちをして再開します。itime を省略すると、ずっと待ち続け、CR キー入力で解除されます。デモプログラムを自動実行させるとき、適度な時間設定をさせる目的に使います。

例: PAUSE 5　　（５秒間、時間待ちをします）

QUIT

Plain_Basic の終了コマンドです。終了確認のダイアログメッセージが出ます。

B4. バッチファイル

解説

Plain_Basic では、インタラクティブモードで、キーボードから入力するそのままを、そっくりテキストファイルに作成しておいて、それを読み込ませて実行させることができます。このテキストファイルを通称でバッチファイルと言います。この方法は、DOS の環境では種々の条件設定のコマンドを処理する方法として使われました。Plain_Basic も、一種の擬似的な環境を持たせていますので、バッチ処理ができます。コンソール入力の手順を書いたテキストファイルを、「DECK ″batch-file″」として読み込ませます。例えば、ファイルの内容が下のように書いてあったとします。まず、prog1file が読み込まれ、リストが出力されて実行され、15 秒の時間待ちをしてから、次の prog2file が読み込まれ、実行されます。この方法での実行は、何かのプログラムのデモンストレーションをさせる目的に使うことができます。

```
LOAD ″prog1file″
LIST
RUN
PAUSE 15
LOAD ″prog2file″
RUN
```

DECK ″batch_file_name″

Plain_Basic のコマンド並びをテキストファイルとしたものをバッチファイルと呼びます。このファイルをを呼んで順に実行させます。

例： DECK ″PROG12″
例： DECK ″B:DEMO.txt″

B5. デバッグ機能

解説

Plain_Basic はインタラクティブなツールですので、プログラムのソースコードから機械語に変換することをしません。したがって、プログラミングと実行時とに簡単なデバッグ機能を設定してあります。その機能は三つあります。

(1) プログラム文にエラーがあると、プログラム文のラベル（行番号）とエラーメッセージを出力します。
(2) コマンド TRON を入力した時点から、実行時の行ラベルをコンソールウインドウに出力します。これによって、プログラムがどの行を実行しているかが分かります。TRON の取り消しは、TROFF を入力します。
(3) バッチ処理の場合、ECHON を入力しておくと、レコードの入力データがコンソールに出力されます。ECHOFF に設定すると、コンソール入力の文字並びもテキストウインドウに出力されなくなります。

ECHON

キーボードからの入力、バッチファイルの入力のエコーリストを、コンソールウインドウに出力します。なお、デフォルトは ECHON に設定してあります。

ECHOFF

ECHON によるエコー出力を中止します。

TRON

プログラムの実行状態を追跡(trace)します。TRON と入力しておくと、実行中のプログラム行の行番号が表示されます。デフォルトは TROFF に設定してあります。

TROFF

TRON によるプログラムの実行状態の追跡を中止します。

B6. 型の定義

解説

変数は定義と宣言とで準備します。定義とは、整数型・実数型・文字型の約束を言い、Plain_Basic 側で決めてあります。これ以外の型の定義、例えば論理型、複素数型などはありません。整数・実数ともに、内部的には倍精度実数（8 バイト）で準備します。４バイト長を使う単精度型実数(DEFSNG)は定義してありません。Plain_Basic で使う文字型は 15 バイトまでの文字列を保存する変数です。ユーザは、変数名を宣言してから使うのですが、明示的に宣言文が必要であるのは配列の場合だけです（DIM 文参照）。単純な変数は、宣言なしに使うことができて、プログラム実行時に最初に変数名が現われたところで、宣言を兼ねてメモリ上に準備されます。その変数名の頭文字を判定して、型が決定されます。この方法は高級なプログラミング言語では許されませんが、数式を扱うプログラミング言語である Fortran では便利に使っていました。また、暗黙（デフォルト）の型定義があります。Plain_Basic も、この定義方式を踏襲しています。文字型の単純変数は、'$'を英数字名の接尾字に使う方法で宣言ができます

DEFINT alpha_charac_1 [- alpha_charac_2]

デフォルトでは、I,J,K,L,M,N で始まる英字名は整数型と定義してありますので、それ以外英字名の名前に適用します。この型の変数は、内部的には小数点以下を持たない倍精度実数として扱います。実数型の変数を代入すると、小数以下を切り捨てます。

例：DEFINT A-C

DEFDBL alpha_charac_1 [- alpha_charac_2]

デフォルトでは、I,J,K,L,M,N で始まる英字名以外は倍精度実数型と定義してあります。

例：DEFDBL M-N

DEFSTR alpha_charac_1 [- alpha_charac_2]

文字型の変数のメモリ上の寸法は、実数型の変数２個分(16 バイト)を使い、15 バイトまでの文字数を保存します。有効な文字の最後にヌル文字 '¥0' が入ります。英数字名の後に'¥'を付けると、頭文字種に関わりなく、明示的に文字型として宣言できます。

例：DEFSTR S

B7. 変数と配列の管理

解説

配列は定義と宣言とで準備します。定義とは、整数型・実数型・文字型の約束を言い、Plain_Basic 側で決めてあります(DEFINT,DEFDBL,DEFSTR)。ユーザは配列名を宣言してから使います。その配列名の頭文字を判定して、型が決定されます。

DIM array-list [,array-list[,....]]

配列を与えられた大きさで宣言します。配列であることを示す添え字は、括弧（ ）ではなく、カギ括弧[ ]でくくります。これは、関数の引き数を表す括弧と区別ができるようにするためです。配列寸法の下限値は１から始めます。C 言語などでは、配列の番号を 0 から始めますが、数値計算の場合には１から始める約束が便利だからです。マトリックスなどは２次元配列で宣言して使います。なお、配列の次元数は４までです。多次元の配列であっても、メモリ上は一次元の並びです。その順番は、例えば M[2,3]の場合には、M[1,1], M[2,1], M[1,2], M[2,2], M[1,3], M[2,3]の順です。

例：DIM A[5],B[5,5]

例：N=6:DIM A[N,N]

ERASE array-list [,array-list[,....]]

メモリ領域を有効に利用するため、不用になった配列または変数を消去します。配列は名前だけで参照します

例：ERASE A,B,C

B8. データ入出力文

解説

Plain_Basic では、外部ファイルを介してデータを直接読み書きする命令文を持たせてありません。しかし、これに代わる方法があります。外部ファイルからデータを読み込むことを擬似的に行わせる方法として、内部ファイルと言う概念を使います。Plain_Basic では、プログラムファイルの中にデータファイルを混ぜて構成します。プログラム文は、各行に整数のラベルを付けます。キーワードの DATA を付けた文は実行文ではなく、データファイルを構成する非実行文です。プログラムの実行時には、DATA 文は読み飛ばされます。READ 文が入力文であって、今度は DATA 文だけを拾い読みして変数に代入します。この処理は、外部ファイルを扱う方法と同質です。

Plain_Basic のプログラム文を作成するとき、実行文だけを集めたプログラムファイルと、DATA 文だけを集めて別のプログラムファイルとを別々に SAVE しておきます。プログラムファイルを LOAD し、DATA 文のファイル MERGE します。幾つかのデータファイルを差し換えて実行させることができます。この場合、プログラム文の整数ラベルが重複しないようにする注意が必要です。

Plain_Basic では、処理結果を二つの子ウインドウ：テキストウインドウとグラフィックスウインドウに出力させますので、このウインドウのデータから外部ファイルに書き出します。この制御は、Plain_Basic のプログラム文ではなく、Windows の画面のプルダウンメニューで行います。

READ variable-list

キーワード DATA が付いたプログラム文だけを選択的に選んで、このデータリストをリストの変数に代入します。この読み出し手順は、ラベルと DATA のキーワードを除けば、外部ファイルからデータを読み込む方法と同じです。

例：READ A,B,C,D （対応する DATA 文は、「DATA 3,4,2.5,6」のように準備します。）

DATA data-list

READ 文で読み込む数値・文字定数のリストを準備しておきます。個々のデータ間の区切り文字(delimiter)は、コンマが標準ですが、スペースで空けても構いません。READ 文のリストに、引き数なしの配列名があるときは、その全要素数のデータリストが必要です。同じ数が何個も必要であるときには、個数 n をデータの頭に付ける表記の約束を決めてあります。また、一定数値での昇順または降順のデータは、FOR-NEXT 文と同様な指定方法を使うことができます。なお、配列名にデータを読み込ませたいとき、配列並びの途中の要素はそのまま残したいことがありますので、その場所を読み飛ばすための記号にスラッシュ'/' を使います。

DATA 文のデータ個数が多くて、一回の READ 文ですべてが読み込まれないときは、次の READ 文が残りのデータを読み込みます。DATA 個数が不足すれば、次の DATA 文に読取り位置が移ってデータ読取りを続行します。DATA 文行末に単独のスラッシュを使うと、READ 文で変数個数が残っていても、そこでデータ読取りを打ち切ります。

例：DATA 1.5, 12, "ABC"　　(コンマ区切りのリスト)
例：DATA 1 2 5 8 9.0 (スペースで区切ったリスト)
例：DATA 10 TO 100 STEP 10　(FOR-NEXT 文と同じ仕様でデータを供給する。個数に注意)
例：DATA 8*0.35　　(同じ数値 0.35 を８個準備する)
例：DATA 10*/,5/　　(元の READ 文の変数個数で 10 個分は読み飛ばす)

RESTORE line-number

READ 文で、データを読み出したい DATA 文の位置を明示的に指定するときに使います。

例：RESTORE 1000

PRINT variable-list

variable-list の値をコンソールに書き出します。テキストウインドウのデータは、メニューの指示で外部ファイルに書き出すことができますので、外部ファイルに直接書き出すことに関連したコマンド（OPEN,CLOSE,PRINT#など）は使いません。数値の出力のときは、デフォルトの書式制御を使います。なお、valiable-list の区切り文字によって、簡単な書式制御のプリントが得られます。

(1) ,(comma) で区切るとリストの間隔はタブで飛ぶ

(2) ;(semicolon) で区切るとすぐ後ろに続いて表示する

例：PRINT "A=";A,X+6

B9. プログラム実行制御

解説

Plain_Basic のプログラムは、RUN コマンドの入力で開始されます。実行は、原則として先頭行からラベルの昇順に行われ、プログラム行が終わったところで直接モードに戻ります。実行順序の変更を制御するプログラム文は４種類あります。IF, FOR, GOSUB, GOTO です。

IF condition THEN statemen-1 [ELSE statement-2]

condition の条件判断を行い、真であれば THEN 以下の statement-1 を実行し、偽であれば ELSE 以下の statement-2 を実行します。condition 部分は論理式と解釈されます。数式であれば、0 が偽(false)、0 以外は真(true)の扱いです。IF 文は文字数が多くなることが多いので、一行に納まる方法の工夫が必要になることがあります。この場合には、GOTO 文を使って別の行に制御を飛ばす方法を使います。マイコンの BASIC 言語では、キーワードの GOTO 無しの書式も許していますが、Plain_Basic では、statement-1/2 共にキーワード名または代入文の変数名で始めなければなりません。

FOR variable = initial TO final [STEP size]::::NEXT [variable]

FOR 文と NEXT 文の間の一連の命令を指定回数だけ繰り返し実行します。STEP の省略値は 1 です。

例：

```
FOR J=1 TO 2
FOR J=1 TO 10                (J のループは I のループの入れ子です)
PRINT A(I,J)
NEXT J
NEXT I
```

GOSUB line-number :::::: RETURN

指定された line-number から始まるサブルーチンをコールし、サブルーチン内の RETURN 文によって、GOSUB 文の次の文へ制御が戻ります。

GOTO line-number

指定された line-number へ無条件で制御が移ります。直接モードで行番号を指定して入力すると、変数領域をクリアしないで実行に入ります。前に実行したときの変数を利用したいときに使うことができます。

B10.　組み込み関数

解説

比較的利用度の高い数学関数を組み込み関数として準備しました。実践的な数値計算に使うために、種類と使い方を限定してあります。三角関数の引き数は、ラジアンではなく度を単位としています。e を底とする自然対数ではなく、10 を底とする常用対数で準備しました。ただし指数関数は e のべき乗です。ベキ関数は、べき乗の演算記号^を使います。したがって、平方根に開くのはその数の 0.5 乗で計算できるのですが、良く使うので SQR 関数を準備しました。なお、余りの計算は、実用計算では利用することが多いので、関数 MOD を用意しました。

ABS(x)

X の符号がマイナスであれば正に直します。例では「 -12.3　　12.3」と得られます。

例：X1=-12.3: X2=ABS(X1): PRINT X1,X2

ATN(x)

逆正接(ARCTANGENT)を度で求めます。角度の範囲は（-180°,180°）です。例では 45 が求まります。

例：Y=ATN(1)

ATN2(ay, bx)

逆正接(ARCTANGENT)を度で求めます。角度の範囲は（-180°,180°）です。例では 180 が求まります。

例：PRINT ATN2(0,-1)

EXP(x)

ｘの指数関数です。例では、2.71828182845905　と得られます。

例：PRINT EXP(1)

LOG(x)

X の符号がマイナスであれば正に直します。例では 2.07918124604762 と得られます。

例：PRINT LOG(120)

MOD(x,y)

x/y の余りを計算します。算術で、「10 割る 3 は、答 3 余り 1」の計算ができます。例では「3.33333333333333 3 1」と得られます。

例：X=10: Y=3: A=X/Y: I=X/Y: PRINT A,I,MOD(X,Y)

RND(n)

区間(0,1) の正規乱数を求めます。引き数ｎはダミーです。

例：E=RND(K)

SGN(x)

引き数ｘの符号を判定して、-1,0,1 を返します。

x<0 の時 SGN(x)=-1.

x=0 の時 SGN(x)= 0.

x>0 の時 SGN(x)= 1.

SIN(x)

ｘの正弦を求めます。角度は度で指定します。範囲は（-180°,180°）です。例題は 0.5 が求まります。

例：PRINT SIN(30)

COS(x)

x の余弦を求めます。角度は度で指定します。範囲は（-180°,180°）です。例題は　0.866025403784439 と求まります。

例：PRINT COS(30)

SQR(x)

x の平方根を計算します。X の符号がマイナスであればエラーを表示します。例では 5 と得られます。

例： PRINT SQR(3^2+4^2)

TAN(x)

x の正接を求めます。角度は度で指定します。範囲は（-180°,180°）です。例題は 1.0 が求まります。

例：PRINT TAN(45)

B11. グラフィックス

解説

コンピュータグラフィックスの利用は、典型的な装置依存のプログラミングです。どのような作図装置を使うかによって、プログラミングの命令語はそれぞれ異なった仕様を持ちます。装置が変っても、プログラム本体の変更が最小で済むように、標準言語の提案が試みられ、また、その標準言語から実際の装置を制御する言語とを繋ぐ考えが生まれました。これをデバイスドライバと言います。ここに挙げたものは、その標準言語の一例です。元々は Fortran のサブルーチン用として利用しましたので、キーワードは英字６文字で表しました。英語の display に関連することを表すため、頭の２字を DP としました。なお、マイコン BASIC では CRT モニタ画面をクリアするときのコマンドに、CLS を使うことがほぼ常識化されています。Plain_Basic ではテキスト用とグラフィックス用の二つの子ウインドウを持ちますので、この消去には CLS と DPERAS と、別々の消去コマンドを使いことにしました。

作図の基本概念は、ペンを使って線を描くことと、ペンの種類を変えることで殆どの図形を描くことができます。ユーザレベルで使い易く設計したプログラムは種々ありますが、Plain_Basic は教育用を念頭に置いていますので、最小限のコマンドに制限しました。線の種類の変更、色の選択、文字フォントの選択は、コマンドで指定するのではなく、主にメニューの方で変更するようにしました。

CLS

引き数はありません。テキストウインドウ（RichTextEdit 領域）のデータを消去します。グラフィックスウインドウの消去は DPERAS です。

DPERAS

引き数はありません。グラフィックスウインドウに作画されている図形を消去します。

DPMOVE x, y

x, y --- 平面ワールド座標系でのペンの位置

グラフィックスウインドウ上に線を引かずに、描き始めのペン位置を設定します。

例：DPMOVE 10.,20.

DPDRAW x, y

x, y --- 平面ワールド座標系での線の終端位置

現在のペン位置から、指定された位置まで、グラフィックスウインドウ上に線を引きます。線の種類は、DPENTX であらかじめ指定します。デフォルトは細い実線です。

例：DPDRAW 12.,15.

DPCIRC x, y, radius

中心位置(x,y)、半径 radius の円を描きます。線の種類は、その前の設定値が使われます。なお、円の内部の塗り潰しは行いません。

例：DPCIRC 0, 0, 50

DPMARK x, y, imark

指定した位置（x, y）に記号を描きます。記号の種類を番号 imark で指定します。寸法はピクセル単位の点です。記号番号と点の図形の対応は、Plain_Basic のバージョンで多少異なります。

例：DPMARK 0,0,3

DPENSZ isize

線の太さを選択します。初期値は１（細線）です。システムの仕様によって、線の太さの選択種は異なります。また、Windows の関数を間接的に利用していますので、線の太さを変えると、DPENTX による破線指定が効かなくなって、すべて実線で描くことがあります。

例：DPENSZ 2

DPENTX ipen

線の種類（実線・破線）などを指定します。初期値は実線(1)です。

例：DPENTX 4

DPTEXT x, y, ″string″

(x,y)を文字列基線の始端として文字を描きます。文字フォントはメニューで指定します。

例：DPTEXT -100, 100, ″test″

DPWIND xcen, ycen, horizontal-size

xcen, ycen --- 画面の中心をこの座標値にセットする

horizontal size --- 画面に納める横幅。デフォルトは 640

グラフィックスウインドウ上の座標系を定義します。グラフィックスウインドウの寸法と縦横寸法比は、マウスによって任意に変化できますが、その中に一杯に接するように作図領域の座標系を設定します。作図領域で考えている縦横縦の寸法比は、使用するスクリーンの寸法比に相似にしておくと納まりがよくなります。デフォルトは 3:4 の比です。この比の変更はメニューで指定できます。一般に、コンピュータグラフィックスの座標指定はモニタ画面の解像度の値を使うことが多いので、デフォルトの横幅寸法を 640 にしました。

例：DPWND 0,0,640

B12. オプション

解説

Plain_Basic のコマンドとステートメントは、初心者教育を考慮して必要最小限に制限しました。しかし、あれば使い易くなる機能を追加する方法を考えることにしました。その方法は、メニューで機能を追加することと、準コマンドとして組み込む方法とがあります。準コマンドは、プログラム文に組み込んでも使うことができます。メニューは、プログラム実行モードでは利用できません。ここでは準コマンドとしての追加仕様であって、OPTION のキーワードに続けて指定の引き数を使います。

OPTION DEGREE

組み込みの三角関数（SIN,COS,TAN）の引き数を度で使いようにします。逆関数 ATN,ATN2 は結果を度で返します。初期設定は DEGREE です。また、Plain_Basic を NEW コマンドで初期化すると設定は DEGREE に戻ります。例えば、「PRINT SIN(30)」は 0.5 が得られます。

OPTION RADIAN

組み込みの三角関数（SIN,COS,TAN）の引き数をラジアンで使いようにします。逆関数 ATN,ATN2 は結果をラジアンで返します。例えば、「PRINT 4*ATN(1)」とすると、3.14159265358979 が得られます。OPTION DEGREE の設定であれば、この場合 180 が得られます。

BM1. TEXT メニュー (Print, ReadFile, SaveFile)

解説

テキストウインドウは、RichTextEdit が載せてあって、これ自体で TxtEditor の機能を持っています。Plain_Basic のテキスト出力がここに書き出されます。この機能を補完するため、内容をプリンタに出力すること、ファイルに保存すること、そしてテキストファイルを読み込むことサブメニューで指示することができます。

Print: あらかじめシステムで定義されたプリンタにテキストを書き出します。プリンタの書式設定なしで書き出しますので、作業時にメモを残すように使います。書式を整える印刷をしたい場合には、一旦 SaveFile で書き出しておいて、ワードプロセッサで編集することを奨めます。

ReadFile: 任意のテキストファイルを読み込んで表示します。この場合、前にあったリストはクリアされます。

SaveFile: 現在表示されているテキストを書式付きテキストファイルに保存します。ファイルの拡張子は(.rtf)であって、rich text format の意です。（なお、拡張子は自動的に付与されませんので拡張子まで正しく描きこむ注意が必要です。）

BM2. GRAPHICS メニュー(CopyToClipboard, PaseteFromClipboard, ReamBmpFile, SaveBmpfile, Print)

解説

グラフィックスウインドウはイメージコンポーネントが載せてあります。ここに作図された図は、クリップボード、ファイル、及びプリンタに出力できるようにしました。作図の例題をデモンストレーションに使うことを考えて、ファイルに保存した図を読み出して表示ができるようにもしました。

CopyToClipboard: グラフィックスウインドウに表示されている図形をクリップボードにコピーします。別のグラフィックスアプリケーションなどで読みだして利用ができます。ただし、このデータはテキストウインドウにペーストできません。Windows システムにはクリップボードビューアがありますので、それを使ってファイルに保存することもできます。

PasteFromClipboard: 上記の CopyToClipboard と対になるように準備しました。

SaveBmpFile: グラフィックスウインドウに表示されている図形をファイルに保存します。このサブメニューをクリックすると、ファイル保存のダイアログボックスが表示され、ファイル名の入力が要求されます。保存するファイルの拡張子は(.bmp)です。（なお、拡張子は自動的に付与されませんので拡張子まで正しく描きこむ注意が必要です。）

ReadBmpFile: 上記の SaveBmpFile と対になるように準備しました。

Print: あらかじめシステムで定義されたプリンタに図形を描きき出します。描き出しは単純な Form の Print メソッドを使いますので、作業時にメモを残すように使います。書式を整える印刷をしたい場合には、一旦 SaveBmpFile で書き出しておいて、別のグラフィックスアプリケーションで編集することを奨めます。

BM3. FONT メニュー (Text, Graphics)

解説

Plain_Basic の初期画面では、文字は９ポイントで表示するようにしてあります。見易さを図るため、テキストウインドウとグラフィックスウインドウそれぞれ別個に文字寸法を変えることができます。s サブメニューをクリックするとフォントダイアログが表示されます。ただし、ここでの指示はポイント数の変更だけしか対応しません。

Text: テキストウインドウの文字寸法を変更します。現在表示されている文字もすべて変ります。

Graphics: この指示以降、コマンドの DPTEXT で表示する文字の寸法を変えます。

BM4. WINDOW メニュー (Vertical, Horizontal, Cascade)

解説

Plain_Basic は、二つの子ウインドウを使い分けることができます。テキストエディタの機能を持つテキストウインドウと、グラフィックスの出力用のグラフィックスウインドウです。タイトルバーには、前者は untitled、後者は Canvas になっています。Basic プログラム文をファイルから読む、またはファイルに書き込むと、untitled は現在のファイル名を表示します。二つの子ウインドウは、親ウインドウの中で位置と寸法を自由に変更できます。Plain_Basic の立上げ時には左右に並べた(Vertical)の状態で表示します。Horizontal, Cascade サブメニューは、他の二つの表示方法を選択します。

Vertical: 左右に子ウインドウを並べます。子ウインドウは相対的に縦長になります。このサブメニュー選択前にアクティブであった子ウインドウが左位置です。テキストウインドウの横幅が狭くなりますので、リストは自動折り返しで表示されますので、見難くなることがあります。

Horizontal: 上下に子ウインドウを並べます。子ウインドウは相対的に横長になります。このサブメニュー選択前にアクティブであった子ウインドウが上位置です。テキストウインドウの横幅が広くないますので、一行の長さが長いリストを見るときに使います。

Cascade: 前後に子ウインドウを並べます。このサブメニュー選択前にアクティブであった子ウインドウが前面になり、後を部分的に隠します。

BM5.　HELP メニュー　(Keywords List, Manual, Version)

解説

Plain_Basic は古典的な CUI の方式にユーザインタフェースを意図的に採用していますので、キーワードを忘れると何もできなくなります。それを補うため、まず、全部のキーワード名の一覧をコンソールにリスト表示するサブメニュー**Keywords List** を付けました。個別のキーワードの説明は、サブメニュー**Manual** をクリックして HtmlHelp で項目を参照できます。その中身は、このマニュアルと同じです。ただし、HtmlHelp は独立した実行形式のプログラムであって、その拡張子は(.chm)です。これは内部的に Microsoft Internet Explorer を呼んでいますので、システムによってはインストールされていないことがあります。

Keywords List: Plain_Basic のキーワードは 60 個程度ですので簡単に覚えられますが、何があって何が無いかの備忘録を兼ねて、全部のキーワードの一覧をコンソールに書き出すようにしました。個々のキーワードの説明は、Manual サブニューを使います。

Manual: このサブニューを初めてクリックすると、Open ダイアログが表示され、そこで HtmlHelp ファイル名を選択することを要請します。Plain_Basic の実行形式のファイルと同じフォルダに(.chm)のファイルがありますので、それを選択します。二度目以降で Manual をクリックすると、直ぐにヘルプファイルが開きます。

Version: この Plain_Basic は種々のバリエーションがありますので、日付に注意します。
