Windows 環境における JIS Full BASIC の実装

白 石 和 夫†

マイクロソフト Windows95 上で動作する JIS Full BASIC 処理系を作成した．この処理系は，数学教育での利用を主な目的として作成したものである．アルゴリズムによる数学的事実の探究を行うための道具として用いられることを意図している．この処理系はボーランド社（現インプライズ）の Delphi 3.1 を用いて作成されている．内部構造は，Delphi のオブジェクトを生成するコンパイラである．若干の非互換を残すものの，JIS Full BASIC 中核機能単位と付属書 I に規定されるモジュールおよび単文字入力の機能単位をほぼ完全に実現している．図形機能単位については，十分ではないが，その主要部分を実現している．極力，規格に合わせる努力をしたが，完全に規格に合致させるのは困難である．規格に合わせるのが難しい点についても報告する．

An Implementation of JIS Full BASIC on MS-Windows

Kazuo Shiraishi†

We developed an implementation of JIS Full BASIC which works on MS-Windows95. This implementaion is designed for use in mathematics education, especially for investigation of mathematical facts. This implementation is compiled with Borland Delphi 3.1. The internal structure is a compiler which generates objects of Delphi. The core module, the modules module and the individual character input module are implemented, while a little incompatibility remains. Furthermore, Major part of the graphics module is implemented. However, Full BASIC standard has difficulty in conformance. The reason why it could not be conformed to is reported on this issue.

1. はじめに

1.1 開発のねらい

筆者は，数学教育での利用を目的として Full BASIC 処理系を開発してきた[1)~4)]．プログラミングの活動が論理の組立てに焦点化されたものとなり，さらには，アルゴリズムによる数学的事実の探究が目的となることを願い，Full BASIC が現実に作成可能で，教育用として有効なものであることを示すのが目的である．

1.2 マイクロソフト互換 BASIC の問題点

現行学習指導要領のもとで，大学入試センターが実施するセンター試験で数学 A，数学 B の内容としてマイクロソフト社の文法で書かれた BASIC プログラムが出題されている．センター試験にマイクロソフト社の文法で書かれたプログラムが出題されるのが常態化すれば（現実にそうなりつつあるが），教育現場ではマイクロソフト社の文法と互換性のある BASIC を使わざるをえなくなる．書かれたプログラムを読む程度であれば，さほど問題とはならないであろうが，生徒自身にプログラムを書かせる教育を行おうとすると，厄介な問題が出てくる．

たとえば，

```
IF 20<=N<30 THEN ……
```

のようなコードを書くと，マイクロソフト社が拡張した文法では比較演算の結果は真偽に対応する数値であるために，`20<=N<30` は数学での通常の用い方と異なる意味になってしまう．そのほか，マルチステートメントや数値の型の別など，マイクロソフト社が独自に拡張した文法は，初心者を悩ます原因になりやすい．

1.3 Full BASIC

JIS Full BASIC[5)]は，ISO Full BASIC，ANSI Full BASIC[6)~8)]とほぼ同一の内容を持つ BASIC 言語の完全バージョンである．JIS Full BASIC には，10 進演算，構造化プログラミング，プログラム分割，配列処理・行列演算，例外状態処理，図形機能などといった特徴がある．

10 進演算は Full BASIC の大きな特徴である．数値を 10 進小数で表現すると，どのような場合に計算結果が正確な値をとるかについて十分な見通しを持っ

† 文教大学教育学部
Faculty of Education, Bunkyo University

て利用できるばかりでなく，実数を有限小数で近似する以上避けることのできない桁落ちの問題なども，コンピュータを実際に利用するなかでそのメカニズムに気づかせることが可能になるなどの利点がある．

10 進演算の結果の正確さに対して厳格な規定が存在することも Full BASIC の特徴である．利用者は，数値があらかじめ定められた有限の桁数で表現されるという制約にさえ注意すれば，コンピュータが行う計算を全面的に信頼してプログラムを書いてよいという環境が用意されることになる．

例外状態処理は，コンピュータを探求のための道具として利用するためには不可欠な機能である．Full BASIC の例外状態処理は，マイクロソフト系 BASIC の例外処理（ON ERROR GOTO）と比べ，構造化されているために扱いやすい．

図形機能は利用者が任意に設定する座標系（問題座標）に基づいて実行される．そのため，関数のグラフを描いたりするような場合に扱いやすくなっている．そして，アフィン変換を利用した図形の変換が簡単な構文で実行できるようになっているので，幾何学の新しい学習手段として期待できる．

Full BASIC では，option 文の arithmetic 選択子に NATIVE を書くことでハードウェア浮動小数点演算を選択することもできる．しかし，1 つのプログラム単位中に複数の型の数値を混在させることはできない．

なお，ANSI Full BASIC は，BASIC 創始者である J.G. Kemeny や T.E. Kurtz らによる True BASIC を土台として設計されたものであるが，True BASIC と Full BASIC の間には無視できない相違がある．たとえば，True BASIC の Windows 版では 10 進演算は実現されていないし，図形機能で細部に相違がある☆．

2. 開発した処理系の概要

2.1 概　　要

本稿では，Web☆☆上で（仮称）十進 BASIC for Windows 95/98/NT4.0 として公開している最新版（ver. 4.55）（**図 1**）に基づいて JIS Full BASIC の実現の可能性について論じる．このバージョンは，マイクロソフト社の OS Windows 95/98 で動作し，CPU が intel x86 系であれば WindowsNT4.0 でも動作する．ただし，コプロセッサ命令で発生する例外の処理を Windows に依存するので，NEC PC9800 版 Windows95 では OS の修正が必要になる．

☆ http://www.truebasic.com/よりダウンロードした Bronze Edition Demo による．

☆☆ http://www.vector.co.jp/authors/VA008683/

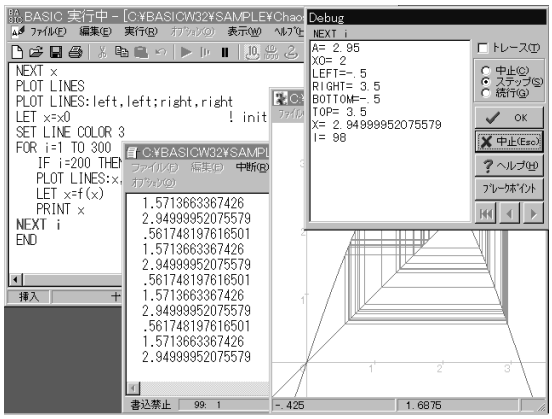

図 1 実行画面
Fig. 1 A screen shot.

この処理系では，若干の非互換を残すものの，JIS Full BASIC 中核機能単位と附属書 I に規定されるモジュールおよび単文字入力の機能をほぼ完全に実現し，図形機能単位の主要部分を実現している．実現されていないのは，内部形式拡充ファイル，固有形式拡充ファイル，実時間，固定小数点 10 進数，編集の各機能単位である．

この処理系は Borland Delphi 3.1 を用いて作成されている．Delphi は C++とよく似た構造を持つオブジェクト指向 Pascal であり，この処理系は，内部的には，Delphi のオブジェクトを生成するコンパイラである．ソースコードの行数はおよそ 45000 行である．

2.2 拡張機能など

この処理系は JIS Full BASIC をできるかぎり忠実に実装することをめざしているが，実用の観点から，いくつかの拡張機能も用意している．

その第 1 は，行番号の省略を許すことである．Full BASIC は規格上，すべての行に行番号を書くことを要求するが，本質的には行番号が不要な制御構造を用意している．探求目的のプログラムを作成する場合などでプログラムの改変が容易にできるよう，行番号は省いてよいものとした．

また，数値については，JIS に対応した十進モード，option 文の arithmetic 選択子に NATIVE を書いた場合に相当する 2 進モードのほか，十進 1000 桁モード，複素数モード，分母，分子がそれぞれ 10000 桁以内の分数の計算が可能な有理数モードを用意した．これらは，option 文で指定することもできるが，メニュー選択によっても変更することができる．十進 1000 桁モードは厳密にいうと 10^9 を底とする演算を行うモードであるが，少なくとも 1000 桁までの 10 進数を正確に扱うことができる．ただし，三角関数や指数・対数

のような超越関数は使えず，べき乗演算のべき指数はある範囲の整数に限定される．複素数モードは，ハードウェア浮動小数点演算を利用して高速に計算を行い，カオスやフラクタルの探究などの目的に使われることを想定している．

2.3 10 進 数

JIS Full BASIC の特徴は，数値に型の別を設けず，数値をすべて 10 進浮動小数点数として扱い，また，演算結果の正確さを厳しく要求することである．この処理系では，10^9 を底とする多倍長の演算ルーチンを用意して対応している．ただし，十分な検討を経ているわけではないので，規格を完全に満たしているという保証はない．

Full BASIC には，数値演算に中間結果という概念がある．数値式や数値関数の計算結果は中間結果である．公称精度を m 桁とする場合には，正確に m 桁の値をとる数値変数のほかに，$m+1$ 桁以上の精度を持つ中間結果が扱えなくてはならない．そして，数値関数などの演算結果の誤差は有効数字の $m+2$ 桁めで 5 を超えてはならず，真の結果が $m+1$ 桁で表現できるときには計算結果は厳密に正確でなければならないと規定されている．この処理系では，数値変数は 15 桁の精度を持つものとしている．したがって，中間結果は，16 桁までの 10 進数が正確に表現可能で，さらにそれを超える桁数の数値が表現できて，演算結果の誤差は 17 桁めで 5 以下であり，計算結果の真の値が 16 桁の 10 進数で表現できる場合には正確な数値となることが要求されることになる．

数値は，内部的には 10^9 を底として表現される．その 1 桁を 32 ビットの整数に割り当てている．桁数を可変としているため，仮数部は，32 ビット整数を不定個数並べたものになる．Delphi のコード上では**図 2** のように表現される．ただし，実際には，技術的な理由で図に示すものとはやや異なる．

この表現が表す数値は，

$$\mathrm{sign} \times (10^9)^{\mathrm{expn}} \sum_{i=1}^{\mathrm{places}} (10^9)^{-i} \mathrm{frac}[i]$$

```
type
  Number = record
     places: LongInt;    //桁数
     sign:   shortint;   //符号
     expn:   smallint;   //指数
     frac:   array[1..116] of LongInt;
  end;
```

図 2 Number 型の定義
Fig. 2 The definition of Number type.

である．places フィールドで桁数を管理することで，必要なサイズのメモリを確保すれば数値が表現できるようになっている．加減乗除の演算ルーチンは，桁数を制御する広域変数を参照して，必要とされる桁数まで計算するように作成してある．

数値変数も実体としては中間結果と同じものである．let 文などを実行して数値変数に中間結果が代入されるとき，有効数字の桁数がちょうど 15 桁になるように四捨五入することで，見かけ上，数値変数の精度が 15 桁となるようにしている．なお，JIS は，数値関数の結果は中間結果であることを要求しているので，数値関数定義 let 文では丸めを行わない．

べき乗演算を含め，無理関数の計算は，原則として，10 進数を数値演算コプロセッサの拡張精度数に変換し，コプロセッサ命令を利用して演算を実行し，再び 10 進数に戻すことで行っている．10 進数に戻す際には，17 桁めの数字が 0，2，4，6，8 のいずれかになるように丸めている．したがって，真の値が 16 桁の 10 進数で表現できるような計算については，丸める前の結果において 17 桁めで誤差が 1 以内となるように評価できれば規格どおりの結果が得られることになる．そして，それ以外の計算の場合は，丸める前の段階で 17 桁めの誤差が 4 を超えていなければ規格を満たすことになる．

拡張精度数は 64 ビットの精度を持つから，10 進数を拡張精度数に変換する際に混入する誤差（相対誤差）は 2^{-64} $(\approx 0.54 \times 10^{-19})$ の程度である．10 進数 x を 2 進数に変換した結果が $x+\Delta x$ になるものとすると，関数 $f(x)$ に対して

$$\frac{f(x+\Delta x)-f(x)}{f(x)} \fallingdotseq \frac{\Delta x}{x}\frac{xf'(x)}{f(x)}$$

となるから，

$$\left|\frac{xf'(x)}{f(x)}\right| \leq \frac{0.4\times 10^{-16}}{2^{-64}} \approx 740$$

となる範囲で相対誤差を 0.4×10^{-16} の程度におさえられる．

たとえば，EXP 関数について考察してみると，$f(x)=e^x$ とおくとき，

$$\frac{xf'(x)}{f(x)} = x$$

であり，また，最大の正数（MAXNUM）が 1E99 であることから EXP(x) の定義域が $\log 10^{99}$ (≈ 228) 以下に制限されるため，実質的な定義域の全体で

$$\left|\frac{xf'(x)}{f(x)}\right| \leq 228$$

が成立し，誤差が 17 桁めで 5 を超えない正確さを保

証する余裕がある．

関数によっては特別な処理を行う場合がある．対数関数，三角関数，逆正弦・逆余弦，べき乗の計算について，その概要を述べる．

$f(x) = \log x$ のとき，

$$\frac{xf'(x)}{f(x)} = \frac{1}{\log x}$$

であるから，x が 1 に近い数であると 10 進数を 2 進数に変換することにともなう誤差を無視できない．現バージョンでは，$0.9 < x < 1.1$ であるときには，10 進数の段階で $h = x - 1$ を計算し，それを 2 進数に変換してから数値演算コプロセッサの命令 FYL2XP1 を用いて作成した $\log(h+1)$ を計算するルーチンに渡して計算している．

三角関数については，演算結果の正確さに関する要求は引数が -2π から 2π までの範囲にある場合に限定されている．裏を返せば，引数が -2π から 2π までの場合には計算結果の正確さを要求するということであり，たとえば，x が π に近い数である場合の SIN (x) の計算結果にも正確さが求められることになる．この処理系では，10 進数の段階で $\pi - x$ の計算を高い精度で行うことで対応している．

逆正弦関数 ASIN (x) では，$|x|$ が 1 に近い数である場合，10 進数を 2 進数に変換する際に生じる誤差が無視できなくなる．そこで，10 進数の段階で $1-x$ と $1+x$ を計算し，これらを 2 進数に変換して，公式

$$\sin^{-1} x = \tan^{-1} \frac{x}{\sqrt{(1-x)(1+x)}}$$

を適用する．逆余弦 ACOS (x) の計算も基本的な方針は同じである．

次にべき乗演算について説明する．$z = x^y$ とすると，

$$\frac{\Delta z}{z} \fallingdotseq \frac{\partial z}{\partial x}\frac{\Delta x}{z} + \frac{\partial z}{\partial y}\frac{\Delta y}{z} = y\frac{\Delta x}{x} + \log x^y \cdot \frac{\Delta y}{y}$$

であるので，単純に x,y を 2 進数に変換して計算したのでは $|y|$ が大きいときには 10 進数を 2 進数に変換する際に生じる誤差が問題になる．$|y|$ が大きい値をとりうるのは x が 1 に近い数の場合であるので，$0.9 < x < 1.1$ のとき，10 進数の段階で $h = x - 1$ の計算を行い，h を 2 進数に変換した後，$x^y = 2^{y\log_2 x} = 2^{y\log_2(1+h)}$ の関係を利用して計算する．この不等式の範囲外では，y を整数部分 n と小数部分 f とに分け，10 進数の段階で乗算の反復をもとに x^n を計算し，それを 2 進数に変換したものに，2 進演算によって x^f を求めた結果を掛ける．

最大の正の数が 1E99，最小の正の数が 1E-99 であることから，$|y|$ の上限を $\log 10^{99}/|\log x|$ で，$|\log x^y|$ の上限を $\log 10^{99}$ で見積もることができる．また，x が 1 に近い数であるとき，$\Delta x \approx |x-1| \cdot 2^{-64}$ であり，さらに，$\log x \fallingdotseq x - 1$ の近似式を用いると，$|\Delta y/y|$ の上限が 2^{-64} で見積もられることを合わせて，

$$\left|\frac{\Delta z}{z}\right| \leq |(\log 10^{99}) \times 2^{-64}| + |(\log 10^{99}) \times 2^{-64}|$$

となる．したがって，x が 1 に近い数である場合に，10 進数を 2 進数に変換したことに起因する相対誤差の上限はおよそ 0.247×10^{-16} と見積もることができる．

$y\log_2(1+h)$ の計算結果は 2^{-64} 程度の相対誤差を持つはずなので，その影響を考察する．$w = y\log_2(1+h)$ に現れる誤差を Δw，$z = 2^w$ に現れる誤差を Δz とすると，$\Delta z/z \fallingdotseq \Delta w \log 2 = (\Delta w/w)\log z$ であり，$|\log z|$ の上限が $\log 10^{99}$ であるので，z に現れる相対誤差の上限をおよそ 0.124×10^{-16} と見積もることができる．h，y を 2 進数に変換する際の誤差と合わせるとおよそ 0.37×10^{-16} で，誤差が 17 桁めで 5 を超えない正確さが実現できているかどうかは微妙なところである．

2.4 内部構造

演算を行うのは（Delphi の）オブジェクトである．たとえば，2 項演算のオブジェクトには 2 個のオブジェクトへのポインタと演算ルーチンへのポインタが埋め込まれている．演算を実行する要求がこのオブジェクトに出されると，このオブジェクトは自身にリンクされている 2 個のオブジェクトに結果を要求し，結果が得られると，リンクされた演算ルーチンを用いて計算を行って結果を要求元に返す．オブジェクトには，変数や定数，単項演算，2 項演算など，いろいろな形をしたものがあり，クラスの継承と仮想メソッドが重要な役割を演じている．

Delphi のコード上では，**図 3** のようにして実現される．TPrincipal は演算を行うオブジェクトの原型（雛型）である．なお，これは抜粋で，実際にはさまざまな機能に対応するメソッドが定義されている．そのようなメソッドの例としては，数値を文字列に変換するルーチンや，反対に文字列を数値に変換するルーチンがある．

実際に演算を行うオブジェクトの例として，2 項演算のクラス型 TBinaryOp を示す．なお，これも抜粋であり，実際には，コンストラクタ，デストラクタをはじめ，種々のメソッドが定義されている．

BinaryOperation 型の手続きの引数は，はじめの 2 つが入力で，最後の 1 つが出力である．number 型の変数を実際には可変長として使っているために，入出

```
type
  TPrincipal=Class(TMyObject)
    procedure evalN(var n:number);virtual;
                                  abstract;
   end;

  BinaryOperation=procedure
           (var a,b:Number; var x:Number);

  TBinaryOp=class(TPrincipal)
         exp1,exp2:TPrincipal;
         opN:BinaryOperation;
    procedure evalN(var n:number);override;
  end;
procedure TBinaryOp.evalN(var n:number);
var
    m:number;
begin
    exp1.evalN(n) ;
    exp2.evalN(m) ;
    opN(n,m,n)
end;
```

図 3 演算を行うクラス
Fig. 3 Classes for operations.

力ともに変数引数となっている．opN フィールドには，翻訳時に加減乗除，べき乗のうちのいずれかを行う手続きが埋め込まれる．

2.5 制　　御

この処理系では，実行時に意味を持つ文は，すべて TStatement と名付けられたクラス型（**図 4** 参照）を継承するクラス型のインスタンス（すなわち，オブジェクト）である．FOR〜NEXT や DO〜LOOP のようなブロックは Full BASIC の構文規則では区と呼ばれるが，print 文や let 文のような単純文も構文規則上，区の仲間である．

TStatement は仮想メソッド exec を持つ．この仮想メソッド exec をオーバライドすることで，文ごとの異なった種類の動作が実現される．各区は，次の区へのポインタ（next）を持っている．区の列（Full BASIC の構文規則上の区*）の末尾の区ではこのポインタは nil 値を持つ．for 区や do 区のように内部に区の列を含む区は，それらの列の先頭の区へのポインタを持っている．if 区の場合には，条件が成立するときに実行される区の列の先頭へのポインタと，条件が成立しないときに実行される区の列の先頭へのポインタの 2 つを持つ．

ExecutiveNext というメソッドはこの区に構文的に引き続く区へのポインタを返す．通常，その値は next の値であるが，next が nil の場合には，この区を末尾に持つ区の列を所有する区の次の区がその値になる．この値を計算するために，実際には TStatement は上記以外のフィールドを持っている．

```
type
   TStatement=class(TMyObject)
      next:      TStatement;
      WhenBlock: TWhenException;
     procedure exec;virtual;
     function ExceptionHandle:boolean;
     function ExecutiveNext:TStatement;
   end;
```

図 4 文のクラス
Fig. 4 The class for statements.

```
var
   CurrentStatement:TStatement;
   NextStatement:TStatement;
function RunBlock
         (statement:TStatement):TStatement;
begin
   result:=nil;
   NextStatement:=statement;
   while NextStatement<>nil do
   try
     while NextStatement<>nil do
      begin
        CurrentStatement:=NextStatement;
        NextStatement:=
                  CurrentStatement.next;
        CurrentStatement.exec;
      end;
   except
      例外を分析;
      with CurrentStatement do
      if (WhenBlock=nil)
      or not ExceptionHandle then
          例外を再生成;
   end;
end;
```

図 5 文を実行する関数
Fig. 5 The function of executing statements.

この処理系には，RunBlock という名前の，区へのポインタを引数とする（Delphi の）関数がある（**図 5** 参照）．この関数の値は区へのポインタであるが，通常は nil を返す．この関数は，引数として指定された区を先頭とする区の列を実行する．RunBlock は，手続き定義と例外処理区から呼び出される．

次に RunBlock の動作について説明する．この処理系には，CurrentStatement，NextStatement という 2 つの静的変数がある．CurrentStatement は現在実行中の区，NextStatement は次に実行される区へのポインタである．RunBlock は引数を受け取るとそれを NextStatement に代入する．そして，NextStatement が nil でなければ，CurrentStatement に NextStatement を代入し，NextStatement に CurrentState-

```
type
   TGOTO=class(Tstatement)
          statement:TStatement;
       procedure exec;override;
    end;
procedure TGOTO.exec;
begin
   NextStatement:=statement;
end;
```

図 6 goto 文のクラス
Fig. 6 The class for goto statements.

ment が指すオブジェクトに含まれる次の区へのポインタを代入した後，CurrentStatement の exec メソッドを呼び出す．この処理は，NextStatement が nil になるまで繰り返される．

この内部構造では，goto 文の実現は容易である（**図 6** 参照）．goto 文の処理を担当するクラスの exec メソッドは，静的変数 NextStatement に次に実行すべき区へのポインタを代入する．

if 区や繰返し区も同様の原理で実現できる．それらの区に属する区の列の最後の文の exec メソッドでは，NextStatement 変数に ExecutiveNext の結果を代入する．

exit-do 文にはそれを取り囲む最も内側の do 区へのポインタが埋め込まれている．通常，実行時には，その do 区に引き続く区へのポインタを NextStatement 変数に代入する．しかし，例外処理区内の exit-do 文で，それを取り囲む最も内側の do 区がその例外処理区が属する保護区を取り囲む場合には，特別な処理が必要になる（後述）．

関数定義や副プログラム呼び出しは，CurrentStatement と NextStatement を退避することで実現される．

2.6 例外状態処理

TStatement のフィールド WhenBlock はその区を when 本体中に含む最も内側の保護区へのポインタである．そのような保護区が存在しなければこのポインタの値は nil である．また，TStatement には，ExceptionHandle という名前の例外状態処理を行うメソッドが定義されている．例外状態処理が正常に終われば この関数の値は真となる．

例外状態処理は RunBlock のレベルで行う．例外が発生すると，RunBlock の例外処理部では，例外を分析して EXTYPE（例外状態の種別番号）を確定した後，CurrentStatement.WhenBlock が nil でなければ，CurrentStatement.ExceptionHandle を呼び出す．ExceptionHandle では，WhenBlock に属する例外処理区が呼び出される．例外処理区が最後まで正常に実行されると NextStatement 変数の値は end-when 行に続く区へのポインタに置き換えられる．

```
function
        TStatement.ExceptionHandle:boolean;
var
  When1:TWhenException;
begin
   result:=false;
   When1:=WhenBlock;
   While not result and (When1<>nil) do
   begin
     try
        When1.RunHandler;
        result:=true;
     except
        on ERetry do
           begin
             NextStatement:=self;
             result:=true
           end ;
        on EContinue do
           begin
             NextStatement:=ExecutiveNext;
             result:=true
           end ;
        on E1:EExitHandler do
             when1:=E1.When.WhenBlock;
     end;
   end;
end;
```

図 7 例外状態処理
Fig. 7 Exception handling.

例外処理区の内側に書かれ，対応する do 区がその例外処理区を含む保護区を取り囲む exit-do 文は，例外を発生する特別な文に翻訳される．RunBolock の実行中にこの例外が発生すると，RunBlock は実行を中断して，対応する do 区の次の区へのポインタを返す．上述の擬似コードではこの処理を省略しているが，例外処理部で NextStatement に nil を代入することで RunBlock を中断させることができる．

この処理系では，retry 文，continue 文，exit-handler 文は Delphi の例外を発生させる文である．ExceptionHandle の実行中に retry，あるいは continue を意味する例外が起こると NextStatement はしかるべき値に書き変えられ，exit-handler を意味する例外が起こると，現在実行中の例外処理区に対応する保護区を取り囲む最も内側の保護区を新たな保護区として例外状態処理をやり直す．それを取り囲む保護区がなくなってしまった場合は，ExceptionHandle は偽となって終了する（**図 7**）．

保護区は TWhenException 型またはそれを継承するクラス型のインスタンスに翻訳される（**図 8** 参照）．

```
type
  TWhenException=class(TStatement)
                                //When-in 区
      block:   TStatement;    //when 本体
      UseBlock:TStatement;    //例外処理区
   procedure exec;override;
   procedure runHandler;
   function ExecHandler:TStatement;virtual;
  end;
procedure TWhenException.exec;
begin
    NextStatement:=Block;
end;
procedure TWhenException.RunHandler;
begin
      nextStatement:=execHandler;
      if NextStatement=nil then
         nextStatement:=next;
end;
function TWhenException.ExecHandler
                                :TStatement;
begin
   result:=RunBlock(UseBlock);
end;
```

図 8 保護区のクラス
Fig. 8 The class for Protection blocks.

TWhenException 型には，その保護区に属する例外処理区を呼び出すメソッド RunHandler が定義されている．

RunHandler では，（仮想メソッド ExecHandler を経由して）例外処理区を実行するために RunBlock が呼び出される．その保護区が when-in 区であるとき，例外処理区の実行中にその when-in 区の外側にある do 区から抜けることを意味する exit-do 文が実行されると RunBlock はその exit-do に対応する do 区の次の区へのポインタを返すので，NextStatement にそのポインタを代入する．それ以外のとき，NextStatement には保護区の次の区へのポインタが代入される．

when-use 区に対応するクラス型は，TwhenException を継承し，handler 区を内部手続きとして呼び出すために ExecHandler をオーバライドしている．

2.7 手続き定義

この処理系では，手続き定義も Delphi のオブジェクトである．Full BASIC の手続きには，関数定義，副プログラム，絵定義，Handler 区がある．関数定義は，実引数をすべて値引数として処理する．一方，副プログラムと絵定義では，実引数はその形によって値引数になる場合と変数引数になる場合とがある．副プログラムと絵定義の違いは，絵定義には実行時に図形変形が指定される可能性があることである．

手続き定義はさまざまな役割を担っているが，変数管理はそのうち最も重要なものである．Full BASIC では，ある副プログラムが複数の call 文から参照される場合，その副プログラム中の仮引数は変数引数，値引数の両様に用いられる可能性がでてくる．そこで，この処理系では，変数はその実領域へのポインタとして管理する．手続き定義が呼び出されると，それらのポインタは別に用意されたスタックに退避される．手続き定義は，呼び出し元から引数リストを受け取る．この引数リストの個々は実引数に対応した（Delphi の）オブジェクトである．仮引数の実領域の確保は，このオブジェクトの仮想メソッドが担う．すなわち，値引数の場合は新たな領域を確保してそこに実引数の計算結果を割り当て，変数引数の場合には，単にその変数へのポインタを返す．手続き定義を抜けるときは，これと逆の操作を行う．

2.8 デバッグ機能

この処理系は教育用に用いられることを目的としているので，誤った論理によるプログラムを実行した場合でも任意の時点で実行を中断できるようにする必要がある．そのため，繰返し文のほか，goto 文や手続き定義など，繰返しを引き起こす可能性のあるオブジェクトを実行するとき一定の回数ごとに（Delphi の）Application.ProcessMessages を呼び出し，利用者からの中断の指示を受け付けるようにしている．また，デバッグ機能の一部としてステップ実行もできるようになっているが，これらは，区の実行時にフラグをチェックすることで実現している（前掲の擬似コードでは省略している）．

2.9 コンパイラ

この処理系は Delphi のオブジェクトを生成する翻訳系であるが，翻訳プログラムの主要部分はそれぞれの文に対応するクラスのコンストラクタとして記述されている．しかし，Full BASIC の構文規則とこの処理系が用意するクラスとが完全に 1 対 1 に対応するわけではないから，この原則に則らない場合も多い．コンストラクタに翻訳プログラムを記述する利点は，コンストラクタで例外が発生するとデストラクタが自動的に呼び出されるようになっているため，構文誤りがあったときに例外を起こすことにすれば，その時点までに生成したオブジェクトの始末が容易になることである．

なお，前掲の擬似コードでは，クラス型の定義において，コンストラクタ，デストラクタ，そのほか，翻訳時に利用するメンバは省いて示している．

3. 規格との相違

3.1 概　　要

この処理系には，若干の非互換が残されている．判明している非互換の全体は配布ファイル内のヘルプファイルに記載してある．本稿では，それらのうち，特に解決が難しいと思われる問題について述べる．

なお，今以上に JIS との整合性を追及するとかえって使いにくいものになってしまいかねない問題もあり，JIS 規格自体の整備・再検討も必要と思われる．

3.2 USING$ 関数など

USING$ 関数における既定の例外状態処理において PRINT USING に対する規定が適用されることになっているが，関数値を書式の長さの星印に置き換えるだけで,「次の行に書式なし出力の表現で値を報告し，…」については省略している．実際にこれを行うのは難しいのではないだろうか．なお，これは，図形 text 文に USING を書いたときも同様である．

3.3 例外状態処理

(1) 例外処理区に書かれた EXIT DO

12.1.4(4) に例外処理区から出る 4 通りの方法が列挙されているが，その中には exit-do 文や exit-for 文が含まれていない．しかし，構文規則では例外処理区に exit-do 文や exit-for 文を書くことは禁止されていない．この処理系では，例外処理区で exit-do 文や exit-for 文を実行した場合には例外処理区から出るものとして扱う．

(2) 例外処理区で起きた例外

12.1.4(8) では，例外処理区の内部で起きた重ねて起きた例外は続行不能なものとして扱うと規定されている．一方，ANSI 規格における対応部分では，例外処理区の内部で起きた例外は省略時想定の例外状態処理手続きで処理されるとなっている．すなわち，ANSI 規格では続行不能例外に対して特別な扱いを要求しているのに対し，JIS では続行可能例外に対して特別な扱いを要求する記述になっている．どちらの立場をとっても不合理であるので，この処理系では，例外処理区で起こる例外に対して特別な扱いをしないものとした．JIS には,「ANSI X3.113 の文章はやや異なるが，TIB によって修正した」とあるが，TIB を入手することができなかったので，記述が変更された経緯は分からない．なお，JIS の「重ねて起きた例外」が exit-handler 文の実行により引き起こされる状態のことを意味するのであれば，この処理系の動作は JIS の記述と合致する．

(3) def 文で起こる例外

12.1.6(9) によれば，def 文の中で例外状態が起こったときに continue 文を実行すると def 行の次の行に制御が移るように読めるが，本処理系では，12.1.4(9) の規定のとおりに動作する．12.1.6(9) の記述は不合理であり，おそらく誤りと思われる．

(4) retry 文

Full BASIC には，retry 文がある．再実行の基準が「文または行」であるので，意味が一意に定まらない場合がある．たとえば，

```
IF a=0 THEN PRINT 1/x
```

で零除算例外が発生して例外処理区に移り，retry 文を実行すると，a=0 の評価から再実行されるのだろうか，それとも print 文から再実行されるのだろうか．JIS の記述はどちらでもいいように読めるので，現バージョンでは行を基準に動作するようになっている．

なお，JIS には「文または行」とあるが,「文」の定義はない．ANSI 規格の対応する部分は “statement or line” となっているので，JIS で「文」となっている部分は単純文（statement）の誤りであろう．

3.4 図形機能単位

Full BASIC の図形機能は 1 世代前のハードウェアを前提としているようで，Windows との相性はかならずしもよくない．しかし，省略時想定の描画領域の形状を正方形とするなどの利用者本位の規定を極力生かす方向で実装している．

この処理系では，Windows のビットマップを規格でいうところの表示装置と解釈することにした．つまり，装置座標空間の原点はビットマップの左下端点であると解釈する．ただし，いまのところ，VIEWPORT，DEVICE VIEWPORT，DEVICE WINDOW，CLIP を質問対象，設定対象とする set 文や ask 文はサポートしていない．

動作の点で深刻な非互換が図形 text 文に存在する．13.3.6(2) の規定では，文字の字形が問題座標系で定義されるように読めるが，この処理系では，字形は問題座標系によらず，Windows の座標によっている．そのため，図形 text 文の出力が問題座標の設定によって縦に伸びたり横に広がったりすることがない．しかし，これを規格に合うように変更すると，文字を出力するときは，座標系を設定しなおすなどの難解なプログラムを書かなければならなくなり，利用者の負担を増大させる結果になる．

3.5 モジュール

JIS 附属書の修飾識別名の構文規則には不備がある．構文規則をそのまま適用するとモジュール名を二重に

含む修飾識別名が生成されてしまう．これは，18.2(26) の構文規則中の，たとえば，数値単純変数名について (27) の構文規則が適用されてしまうからである．今回作成した処理系では，モジュール名を二重に含むような修飾識別名を禁止している．

18.4(1) には，「モジュール本体の public 文で宣言された手続き，変数および配列が最初に参照されるより前に，モジュール本体中の区が実行される」となっているが，これは実行できない場合がある．たとえば，2 つのモジュールで public 文に書かれた変数が相互に相手方のモジュールのモジュール本体で外部共有宣言されている場合はどちらのモジュール本体から先に実行されるのだろうか．これを正しく処理することは不可能と思われたので，今回作成した処理系では，モジュール本体は出現順に実行されるものとしている．

18.2(7) によると，public 宣言や share 宣言は option 文や module-option 文の前に書くことになる．しかし，今回作成した処理系では，base 選択子を持つ option 文はすべての配列宣言より手前，arithmetic 選択子を持つ option 文はすべての数値識別名より手前に書かなければ翻訳できない．これは内部処理の都合であり，いずれは JIS の規定どおりに書かれたプログラムも翻訳できるようにするつもりではあるが，たとえば，

```
110 SHARE NUMERIC A(10)
120 MODULE OPTION BASE 0
130 DECLARE NUMERIC B(10)
```

の順に書かなければならないという JIS の規定にはかなり違和感がある．

モジュール本体区には区が書けるから data 文を書くこともできる．しかし，データ列の有効範囲に関する規定が欠けている．今回作成した処理系ではデータ列の有効範囲はモジュール本体であるものとしている．

モジュール本体区に data 文が書かれうることを想定していないことや 18.2(8) にわざわざ dim 行を加えていることなどを考えると，18.2.(32) に現れる “宣言文”（declaration-statement）は declare 文の誤りで，18.2(8) の意図は区とある部分を区から宣言文を除いたものにすることであったように感じられる．しかし，18.4(6) の規定はこの解釈と矛盾する．

4. 評　　価

4.1 実行速度

実行速度についてテストした結果を示す．以下に示す数値は，CPU に MMX ペンティアム 200 MHz を採用する AT 互換機で動作させた結果である（OS は Windows95）．なお，括弧内に，CPU に AMD K6 266 MHz を採用する AT 互換機での実行結果を付記する（OS は Windows95）．

```
DECLARE EXTERNAL FUNCTION f
LET t0=TIME
FOR n=1 TO 10000
   LET  m=f(n)
   if n<m AND n=f(m) THEN PRINT n,m
NEXT n
PRINT TIME-t0
END
EXTERNAL FUNCTION f(n)
LET s=1
FOR i=2 TO SQR(n)
   IF MOD(n,i)=0 THEN
      IF i=n/i THEN
         LET s=s+i
      ELSE
         LET s=s+i+(n/i)
      END IF
   END IF
NEXT i
LET f=s
END FUNCTION
```

図 9　テストプログラム
Fig. 9　The test program.

2 数 m，n について，m の自分自身を除く約数の和が n で，n の自分自身を除く約数の和が m となるとき，m と n は友数であるという．**図 9** に示すのは，10000 以下の範囲で友数を探すプログラムである．

このプログラムの実行にかかる時間は，十進モードで 11.42 秒（7.08 秒），2 進モードで 3.02 秒（2.96 秒）であった．なお，実行に要する時間は計測のたびごとに変動するので正確な数字ではない．

このプログラムは True BASIC でもそのまま実行可能であり，True BASIC Bronze Edition Demo で実行すると，7.86 秒（5.27 秒）かかった．そのほか，両 BASIC に共通な文法で書かれたプログラムで実行にかかる時間を比較してみると，手続き定義や図形機能を利用するプログラムでは今回作成した処理系のほうが速く，エラトステネスの篩のように初歩的な命令のみを利用するプログラムでは True BASIC のほうが速いという傾向が見られた．

上記のプログラムを Microsoft Excel97 VBA で実行可能となるように最小限の修正を施して実行してみたところ，5.82 秒（5.42 秒）という結果が得られた．修正点は，`PRINT` を `Debug.Print` に，`MOD(n,i)` を `n-i*Int(n/i)` に，`TIME` を `Timer` に書き換え，主プログラムをボタンクリックに応じるプロシージャに変更し，機能語 `EXTERNAL` を削除したことである．

4.2 10 進数演算

十進 1000 桁モードでの結果と比較することで，べき乗演算の誤差を調べてみた．CPU が MMX ペンティアムの場合には良好な結果が得られるが，CPU が AMD K6 の場合にはやや誤差が大きい．たとえば，`0.9900000000001^(-22214)` の正しい値の有効数字の 17 桁め以降は 4782…であるが，MMX ペンティアムでは 17 桁めの数字が 4 となるのに対し，AMD K6 では 2 となる．同様の例が多数見つかるので，計算結果の正確さを保証するためには，CPU 自体の誤差を知っておくことが不可欠だといえる．

5. おわりに

5.1 開発のねらいに関連して

もともとのねらいは，JIS Full BASIC の実現可能性を示すことで，より本格的な Full BASIC 処理系の出現を促すことであったが，いつのまにか自身で本格的な Full BASIC を作成することになってしまった．しかし，完全な Full BASIC の実現までにはまだかなりの距離がある．

この BASIC をインターネット上で公開して以来，多くの方からバグ報告をいただき，処理系を完全なものに近づけるのに役だったが，その多くは，企業の研究開発部門に属すると思われる方からのものである．

最近では，大学，大学院，高専などでもこの BASIC が利用されるようになってきている．しかし，高等学校での利用では，札幌稲北高校の早苗雅史氏☆や立命館高校の文田明良氏ら☆☆の実践など優れたものがあるものの，全般的には低調である．その主因はセンター試験の出題にあると考えられるが，その裏には，構造化プログラミング=システム開発と考える誤解があるように思える．平成 15 年度入学生から実施される高等学校新学習指導要領では，数学 B の内容に「数値計算とコンピュータ」が含まれるが，今春発行された高等学校学習指導要領解説数学編（文部省）には，「……，ここでのプログラミングは，簡単な数値計算のアルゴリズムを理解し作成することを主なねらいとしているので，構造化プログラミングなどについて深入りはしない」（p.97）などと，アルゴリズム記述と構造化プログラミング言語との関係を否定するような記述がある．今後も，このような意識を変えていく努力が必要であろう．

☆ http://www.nikonet.or.jp/spring/sanae/
☆☆ http://www.ritsumei.ac.jp/fkc/ fumita/

参考文献

1) 白石和夫：数学教育で用いるための BASIC 処理系の試作，文教大学教育学部紀要，第 27 集，pp.43–51 (1993).
2) 白石和夫：Windows 環境で動作する BASIC 言語システム，文教大学教育学部紀要，第 30 集，pp.34–40 (1996).
3) 白石和夫：JIS 準拠 BASIC 言語処理系の開発とその目的，日本数学教育学会誌，Vol.80, No.5, pp.82–89 (1998).
4) Shiraishi, K.: Development of a BASIC Language Processing System in Accordance with ISO and its Aim, *Proc. Third Asian Technology Conference in Mathematics*, pp.225–232, Springer (1998)
5) 日本工業規格：電子計算機プログラム言語 Full BASIC，日本規格協会 JIS X 3003-1993.
6) Information Technology Industry Council: American National Standard for Information Systems – Programming Laguages – Full BASIC, ANSI X3.113-1987.
7) Information Technology Industry Council: American National Standard for Information Systems – Programming Laguages – modules and indivisual character input for Full BASIC, ANSI X3.113a-1989.
8) 西村恕彦ほか（編）：アメリカ規格 Full BASIC，共立出版 (1990).

(平成 12 年 3 月 7 日受付)
(平成 12 年 5 月 24 日採録)

白石 和夫（正会員）

1953 年生．1976 年東京教育大学理学部数学科卒業．1981 年筑波大学大学院博士課程数学研究科単位取得退学．都立高校教員を経て文教大学へ．現在，文教大学教育学部教授．アルゴリズムの実行はコンピュータにまかせればよい時代の数学科のカリキュラムの再構築を研究テーマとしている．日本数学教育学会出版部幹事（YEARBOOK 第 4 号編集主任）．