Full BASICのObject Pascalへの埋め込み

白石　和夫*

An Embedding of Full BASIC into Object Pascal

Kazuo SHIRAISHI

要旨　Full BASICプログラムをObject Pascalのプログラムに翻訳することで高速に実行する処理系を作成した．日本工業規格Full BASICに定める図形機能単位および附属書Ⅰに定めるモジュールおよび単文字入力の規格に従って作成され，各プログラム単位にOPTION ARITHMETIC NATIVEを書いたプログラムが対象である．実用上，ほとんどのプログラムが実行可能であるが，理論上は変換できないプログラムも存在する．また，動作上の微細な非互換も存在する．変換先のObject Pascal言語は，Delphi互換のfpc＋Lazarusである．

キーワード：Full BASIC　Object Pascal　translator　BASIC Accelerator　数学探究

１．研究のねらい

数学教育においては，より多くの具体事例に基づいた探究活動の充実が求められるようになってきた．そこでは，コンピュータは不可欠な道具となる．筆者は，Windows上で動作するFull BASIC処理系（図形機能単位および単文字入力・モジュール機能単位準拠）を作成し[1]，公開してきた（(仮称）十進BASIC[2]）．Delphiと互換性のあるPascal処理系FPC＋Lazarusが実用上の完成に近づいたのを受け，MAC（Intel）およびLinux（i386）でも動作するようにした．

Full BASICには，各プログラム単位にOPTION ARITHMETIC NATIVEを書くことで，必ずしも十進演算とはかぎらないハードウェア浮動小数点演算を行わせる道が用意されている．（仮称）十進BASICでもFPUを利用した2進演算をサポートしているが，内部構造はObject Pascalのオブジェクトを生成する翻訳系であり，i386の機械語を生成するネイティブコードコンパイラに比べると実行速度的に不満がある．

そこで，BASICプログラムをObject Pascalに翻訳することで，Pascal処理系を介して高速実行することを目指すことにした．

２．概　要

2.1. 翻訳対象のプログラム

Object Pascalへの翻訳対象となるBASICプログラムは，日本工業規格Full BASIC（JIS X3003）[3]の図形機能単位および附属書Ⅰに規定される単文字入力およびモジュール機能単位に合致するプログラムのうち，各プログラム単位にOPTION ARITHMETIC NATIVEを書いてハードウェアによる数値演算を許すプログラムである．

2.2. 変換先のObject Pascal

変換先のObject Pascalは，ボーランド社のDelphiによって導入されたものに準拠する．Object Pascalには，今のところ，C++のような公的な標準は存在しない．ただし，実行にはDelphiと互換性のあるフリーのPascal処理系fpc[4]とfpc上のGUIライブラリ群であるLazarus[5]を用いる．また，グラフィ

*しらいし　かずお　文教大学教育学部学校教育課程数学専修

ックスや，テキスト出力の画面表示，INPUT文実行時の画面からの入力などにはLazarusが提供する枠組みを用いて作成したライブラリを用いる．Full BASICの文や関数は，基本的に，対応する機能を持つPascalの手続き・関数を用意することで対応する．

３．手続き定義

3.1. 外部手続き定義と内部手続き定義

Full BASICのプログラムは，プログラム単位を並べたものである．プログラム単位は，プログラムの最初に書かれる主プログラムと，それに続く外部手続き定義である．

Full BASICでは，プログラム単位の内側に内部手続き定義が書かれる．CやJavaにはFull BASICの内部手続きに対応する概念がない．一方，Pascalは，手続き定義のネスティングを許す．だから，内部手続き定義を含むFull BASICプログラムの変換先としては，事実上，Pascal以外には考えられない．

3.2. 引数の参照渡し

Full BASICの副プログラムと絵定義は，実引数に変数を書くとその変数は参照渡しとなる．一方，実引数に変数として解釈できない式を書くと値渡しになる．Pascalの手続きにも参照渡し，値渡しの区別が存在するが，それは仮引数の属性で，一つの仮引数を参照渡しにも値渡しにも使えるものではない．

副プログラム自体はすべて参照渡しで定義しておき，手続き呼び出しごとに，値渡しがあれば追加の変数を用意して手続き呼び出しの前にその値を代入しておき，手続きにはその変数を渡すことが考えられる．しかし，それでは，実引数に配列要素が混在している場合などで実引数の評価の順序がくるってしまう．その問題を解消するために，変数と値を引数とし，変数へのポインタを返す関数を用意した．この関数は，値を変数に代入し，変数のアドレスを値として返す．値渡しになる実引数には，この関数を逆参照したものを書く．

3.3. 関数定義LET文

関数定義LET文は，文法上，それと対応するPascalの文に書き換える．しかし，Object Pascalでは関数名は局所変数であって関数呼び出しごとに局所的である．一方，Full BASIC規格は，その値が呼出しに対して局所的であることを要求していない．そのため，非互換が発生する．

なお，この値を大域変数にするのは論理的には好ましくないので，非互換を承知の上でそのままにしている．

3.4. モジュール

JIS Full BASIC附属書Ⅰに規定されるモジュールは，外部手続きと広域変数の集まりである．Full BASICのモジュールはObject Pascalの静的オブジェクトに変換する．

４．制御構造

4.1. 論理式

Delphiおよびfpcでは，and，orの短絡評価が可能である．これは，Full BASICのAND，ORの動作と一致するので論理式の変換はさほど難しくない．ただし，Pascalは論理演算の優先順位が算術演算の乗算と同レベルであるので，プログラムの書き換えに際し，括弧を補う必要がある．

4.2. SELECT CASE構文

SELECT CASE構文は，SELECT CASE行に書かれた式の値を保持するための変数を別に確保してif文に書き換えて変換する．

4.3. FOR～NEXT

FOR～NEXTは増分に小数を許すなど，Pascalのfor文にない機能がある．FOR～NEXT構文は，Full BASIC規格におけるFOR区の定義に忠実に，そのfor区でのみ用いられる変数を2個用意し，Pascalのwhile文に書き換える．

4.4. DO～LOOP

DO行かLOOP行の一方にのみ脱出条件を書いたDO～LOOPはwhile文かrepeat ～ until構文に書

き換える．DO行とLOOP行の双方が脱出条件を持つ場合は，while文に書き換えて，ループの最後にif … then break; を挿入する．

4.5. EXIT FORとEXIT DO

Object Pascalの break文がFull BASICのEXIT DOおよびEXIT FORに対応するが, break文はwhile文, repeat ～ until，for文に共通であるので，FOR～NEXTとDO～LOOPが入れ子になっている場合にはbreak文を使うことができない．そのため，ラベルを定義してgoto文に書き換える．

しかし, EXIT文が例外処理区の本体（when本体）に囲まれ，FOR～NEXTまたはDO～LOOPがその例外処理区を含むような場合, Object Pascalに変換したときそのgoto文は許されない．その場合，脱出すべき繰り返し構造をtry ～ except endで囲み，EXIT DOあるいはEXIT FOR専用に定義した例外を発生させる．繰り返し構造が含む例外状態処理もtry ～ except endに書き換えるが, except ～ endのなかでその例外に対しては例外が再生成されるようにする．

4.6. EXIT SUB，EXIT FUNCTION

Full BASICのEXIT SUB，EXIT FUNCTIONに対応するObject Pascalの命令はexit文であるが，Full BASICでは，たとえば，外部関数定義の内側に書かれた内部副プログラム中でEXIT FUNCTIONを実行することも可能で，そのような場合，Pascalのexit文では対応できない．そこで，上述と同様の手法で例外を利用する．

4.7. GOSUB ～ RETURN

Full BASIC規格は，GOSUB文の実行回数とRETURN文の実行回数が一致しないプログラムを認めている. だから, GOSUB文をアセンブラのcall命令に変換することはできない.

規格が定める通りに各プログラム単位に棚を用意し，GOSUB文を実行するとき，戻るべき行の行番号を棚に積む．RETURN文は，棚から行番号を取り出し，その番号をcase文で選択してgoto文で戻る. GOSUB文の所在は処理系がすべてを把握しているから，case文には可能性のある行番号に対応するgoto文を羅列しておく．実行速度的には好ましくないが，Full BASICではGOSUB～RETURNを使わないことが推奨されているから大きな問題ではない.

5．例外状態処理

5.1. WHEN EXCEPTION IN ～ END WHEN

FULL BASICのWHEN EXCEPTION IN ～ USE ～ END WHENは, Object Pascalのtry ～ except ～ endに対応する．Full BASICのEXIT HANDLER文は, Object Pascalの引数を持たないraise文に相当する．

5.2. 数値演算

Full BASICは，CやJavaと異なり，桁あふれや零除算など，数値演算の誤りで例外を発生させる言語である. Object PascalではFPUの例外フラッグを有効にして数値演算の誤りで例外を発生させるのが通例であり，Object PascalはFull BASICと相性がよい．しかし，Full BASICの例外をObject Pascalで再現するのは難しい.

5.3. 例外番号

Object Pascalの例外状態は1つのオブジェクトであるのに対し，Full BASICの例外状態は例外番号（EXTYPE）で識別される．Full BASICでは例外番号が演算の桁あふれや算術計算の誤り（零除算など）に対して細かく分類されているが，Object Pascalの例外オブジェクトにはおおまかな分類しか存在しない．そのため，正確な例外番号を生成するために，個々の演算ごとに例外が発生したときの例外番号をあらかじめ指定しておかなければならない．しかし，これは，実行速度の低下を招く．組込み関数の実行時にのみその処理を行い，四則演算ではObject Pascalの例外を発生させ，EXTYPE関数のなかでBASICの例外番号に変換するようにする.

5.4. 例外状態の伝達

Full BASICは，手続きの実行中に例外が発生し，それを取り巻くwhen本体がないか，または，例外

処理がEXIT HANDLERの実行によって取り消されたかすると,例外状態を呼び出し元に伝達する.メカニズム的にはObject Pascalの例外処理も同等の機能をもつが,Full BASICでは例外番号(extype)の値に100000を加算する処理が要求される.そのため,各手続きの内側にtry ～ except endを持たせ,例外発生時に例外番号を修正する.

5.5. 行番号分岐

Full BASIC規格は,次のようにwhen本体内からwhen-in区の外へ行番号分岐によって分岐することを禁止していない.

```
100 WHEN EXCEPTION IN
110     GOTO 140
120 USE
130 END WHEN
140 END
```

しかし,Object Pascalではtry ～ exceptからgoto文で抜けることができない.そのため,上のようなプログラムは変換できない.

この事情から既に述べたように,when本体中に書かれたEXIT DO文,EXIT FOR文はgoto文でなく例外の発生に変換する.その例外を受けるためにDO～LOOPまたはFOR～NEXTはtry～exceptで囲まれる.そのため,保護区を含み,その保護区内にEXIT DO文を持つDO区の内から外への分岐と,保護区を含み,その保護区内にEXIT FOR文を持つFOR区の内から外への分岐ができなってしまう.たとえば,次のプログラムは変換して実行できない.

```
10 DO
20    WHEN EXCEPTION IN
30       EXIT DO
40    USE
50    END WHEN
60    GOTO 80
70 LOOP
80 END
```

上に述べた事情は,さらに次のような非互換ももたらす.すなわち,WHEN-IN区に属する例外処理区がGOSUB～RETURNを持つと,いずれの保護区にも属さないGOSUB～RETURNを書くことができない.

5.6. RETRY文とCONTINUE文

RETRY文とCONTINUE文と同等の機能をObject Pascalで実現するのは難しい.Object Pascal自体には,例外発生文を再実行させる命令は用意されていないし,仮にそれが可能であったとしても,変換元のBASICの文と変換先のPascal文とが1対1に対応するとは限らない.

RETRY文またはCONTINUE文を例外処理区に含む保護区では,when本体の各文の実行前に行を識別するための番号を特別な変数に記憶させる.そして,RETRYまたはCONTINUEが実行されると,when本体の最初に戻り,case文でその数値を調べてgoto文で適切な位置に飛ぶ.

たとえば,

```
WHEN EXCEPTION IN
   FOR x=-4 TO 4 STEP 0.01
      PRINT x,1/x
   NEXT x
USE
   CONTINUE
END WHEN
```

を変換すると,次のようなプログラムに変換される(変数宣言などは省略).

```
// WHEN EXCEPTION IN
Continue:=0;
repeat
try
Case Continue of
0: ;
1:begin continue:=0; goto a1 end;
2:begin continue:=0; goto a2 end;
3:begin continue:=0; goto a3 end;
4:begin continue:=0; goto a4 end;
end;
//    FOR x=-4 TO 4 STEP 0.01
```

```
ExLineNumb:=1;
__own1_00000 := 4 ;
__own2_00001 := 0.01 ;
_X := -4 ;
while sign(_X - __own1_00000)
          * sign(__own2_00001)<=0 do
begin
//       PRINT x,1/x
ExLineNumb:=2;
console.PRINT([],rsNone, true ,
                [_X, TNewZone.create ,
                1/(_X), TNewLine.create ]);
a2:
/    NEXT x
ExLineNumb:=3;
_X := _X + __own2_00001 ;
a3:
end;
a1:
// USE
a4:
except
 on E:EControlException do raise;
 on E:Exception do begin
  base.extype:=0;
  ExcodeRec:=Excode;
  Excode:=DefaultExcode;
//    CONTINUE
Continue:=ExLineNumb;
goto h5;
// END WHEN
h5:
 end;
end;
until Continue=0;
```

通常は実行されることのない代入文を各行ごとに実行するので，速度は低下する．また，実際にRETRYまたはCONTINUEを実行する場合には，case文に羅列された項目を上から順にテストして飛び先を決めるから，さらに動作が遅くなる．なお，Pascal処理系がラベルをアドレスに変換する機能を持っていれば，例外発生時の例外オブジェクトの例外アドレスと照合することで例外発生行が特定できるので，各文ごとにExLineNumb変数に行番号を代入する処理は不要になる．

６．変数および代入・入出力

6.1. 配列

Full BASICの配列は，それ自身が添え字の上限，下限を管理している．また，下限は任意の整数に設定することができる．そのため，Full BASICの配列は，Object Pascalのオブジェクトに変換する．

6.2. 代入

Full BASICは，次に示すプログラムの30行のような特別な形式の代入文を持つ．

```
10 DIM A(4)
20 LET N=2
30 LET N, A(N)=4
40 MAT PRINT A
50 END
```

この代入は並行的に行われる．C言語の複代入文のように代入文の結果を次の代入文の入力とする方法では駄目で，まず，変数のアドレスを求め，右辺の値を計算した後，その値をさきほど定めたアドレスにある変数に代入する方法によらなければならない．それを実現するために，不定長のアドレス配列と代入すべき値を引数とする手続きを定義する．

```
procedure LET(const p:Array of PDouble;
                const x:double);overload;
var   i:integer;
begin
   for i:=0 to High(p) do
       p[i]^:=x;
end;
```

なお，この手続きがoverloadで宣言されているの

は，文字列変数用も用意するためである．

6.3. 文字列変数

Full BASICの文字列変数には部分文字列指定を含むことができる．文法上，文字列変数名を書くことができるところには，部分文字列指定を持つ文字列変数を書くことができる．

文字列変数に対する多重代入文を変換するためには，部分文字列指定を持つ文字列変数と持たない文字列変数を同列に扱うメカニズムが必要になる．そこで，文字列変数に対するLET文は，次のように定義された型TStrVarの配列を引数とする手続きとして用意する．TStrVar2をTStrVarを継承する型として定義することで，TStrVar型の引数のところにTStrVar2型のオブジェクトを代入することができる．なお，TStrVar2のメンバー変数left, rightは部分文字列指定の2数を意味する．

```
type
 TStrVar=class
    PVar:PString;
   constructor create (P:PString);
   procedure setstring(const s:string);
                                  virtual;
   function getstring:string;virtual;
   property str:string read getstring
                     write setstring;
 end;
 TStrVar2=class(TStrVar)
    left,right:integer;
   constructor create (P:PString;
                    l,r:integer); overload;
   constructor create (P:PString;
                  l,r:double); overload;
   procedure setstring(const s:string);
                                override;
   function  getstring:string;override;
end;
```

すると，文字列変数の多重代入を行うLET手続きを次のように定義することができる．

```
procedure LET(const p:Array of TStrVar;
                const s:string);overload;
var   i:integer;
begin
   for i:=0 to High(p) do
      begin
         p[i].str:=s;
         p[i].free
      end;
end;
```

6.4. INPUT文

Full BASICのINPUT文は，複数の入力項目を書いたとき，左に書かれた変数に入力がなされてから配列添字や部分文字列指定が評価される．たとえば，

```
100 INPUT N, A(N)
```

を実行するとき，配列要素が定まるのは左にあるNに値が入力された後である．そのため，あらかじめ変数のアドレスを取得しておいて，入力を読みながら順に代入していく手法は取れない．

規格では，左の変数から順に，入力の構文的な正しさを確認した後，その値が代入されることになっているが，これは難しい．現状では，入力すべき変数の個数分のデータ要素が揃い，行全体の構文的な正しさがすべて確認されてから順に代入している．そのため，入力を途中までにとどめた場合には非互換が生じる．

6.5. PRINT文とPRINT USING文

Full BASICのPRINT USING文では，書式文字列は文字列式で与えられるから，翻訳時には確定していない．すなわち，書式の適用は実行時に行われる．また，PRINT文も含め，印字項目の個数も不定である．そのため，対応するPascalの手続きを型可変オープン配列を引数とする手続きとして用意し，翻訳時にはオープン配列コンストラクタを利用して引数を作る．

6.6. DATA文

Full BASICのDATA文は，プログラム単位ごとに独立であり，さらに，そのデータ列は外部手続きの呼び出しに対し局所的である．だから，DATA

文を含むプログラム単位は，DATA列へのポインタを局所変数として含まなければならない．これは，DATA列をオブジェクトとして定義し，READ文やRESTORE文をそのメソッドとして記述し，各プログラム単位にオブジェクト型の変数を用意してその実行開始直後にインスタンスを生成する方法で解決する．

6.7. ファイル

Full BASICでは，ファイル入出力は経路を介して行われる．経路は数値で識別されるが，経路番号は数値式で書かれるから，経路番号が確定するのは，実行時である．経路自体は大域的な存在であるが，経路番号の有効範囲はプログラム単位である．また，経路は外部副プログラムの引数にもなる．そのため，各プログラム単位には，経路番号と経路の実体との対応を管理する機構が必要になる．

経路自体は大域的なオブジェクトとして，また，入出力はそのメソッドとして実装する．ファイル入出力は，ファイル番号式を評価し，それが指し示す経路オブジェクトが行う形になる．

すなわち，各プログラム単位は，経路番号を管理し，OPEN文，CLOSE文を処理するオブジェクトを持つ．OPEN文は，ファイルの実体を意味する大域オブジェクトを生成する．

Full BASIC（中核）のファイルには，表示形式，順編成内部形式，流れ編成内部形式の別がある．それらは，基底型のオブジェクトを継承するオブジェクトとして実装する．

6.8. IF MISSING THEN

Full BASICには，他の多くの言語にあるようなEOF関数は存在せず，入力文にIF MISSING句を書いてデータ要素が尽きたときの処理を指定する．そのため，入力文の翻訳はかなり面倒である．たとえば，

```
OPEN #1: NAME "a:ABC.TXT"
DO
   INPUT #1,IF MISSING THEN EXIT DO: a,b
LOOP
CLOSE #1
```

をPascalに翻訳すると，

```
ChannelList.Open(1,'a:ABC.TXT','OUTIN',
    'DISPLAY','SEQUENTIAL',maxint, false );
repeat
with TDataList.create do
try
 Missing:=false;
if InputData(ChannelList.channel(1),'nn',[],
'?',MaxNumberDouble,nil,false,rsNone,2) then
begin
 _A := FloatVal( strings[0]) ;
 _B := FloatVal( strings[1]) ;
end else
 Missing:=true;
finally
 free;
end;
if Missing then
           begin base.extype:=0; goto L1;end;
until false;
L1:
ChannelList.close(1);
```

のようになる．ChannelListは，プログラム単位ごとに存在する経路管理オブジェクトである．ChannelList.channel ()は，指定の経路番号で開いた経路オブジェクトを指し示す関数である．missingはプログラム単位で宣言されたブール型の変数である．TDataListは，TStringList を継承して定義されている．TDataList型のメソッド関数InputDataは，データ列が尽きたときには偽を返し，それ以外の要因でエラーになったときには，Full BASICの例外番号に対応した例外を生成し，正常に動作を終えたときには，strings配列に入力された値を保持している．InputDataの2個目の引数'nn'は，数値2個を入力させる指示である．FloatVal関数はBASICの数値リテラルをDouble型の数値に変換する．

7．図形機能

7.1. GRAPH文・PLOT文

Full BASICのGRAPH文，PLOT文は不定個数の引数を持つ．Pascalでは，オープン配列パラメータを持つ手続きとして用意し，呼び出し時にオープン配列コンストラクタを利用する．

7.2. 図形変形

Full BASICの図形機能の特徴は，4行4列の行列を用いた射影変換にある．

PLOT POINTS，PLOT LINES，PLOT AREA，PLOT TEXTのようにPLOTで始まる描画命令を実行すると，現在変形行列を用いて射影変換が適用される．

絵定義と呼ばれる副プログラムの変種があり，組込みの変形関数または4行4列の行列またはそれらの積を指定して呼び出すとそれらが現在変形行列に乗じられる．変形行列を伴う絵定義の呼び出しを多重に行うことも許されているから，現在変形行列は棚構造を持つ必要がある．

Object Pascalに絵定義に対応する機能が存在しないから，絵定義自体はPascalの手続きに変換する．絵定義の呼び出しは，現在変形行列に指定された変形を乗じてスタックを棚に積むコードと，呼び出し本体，そして，棚を元に戻すコードとに分割して生成する．

8．結　語

8.1. 成果

翻訳を行うプログラムは，（仮称）十進BASICの処理系を改変して作成した．字句解析の結果生成されたオブジェクトが実行の代わりにPascalコードを生成する形になっている．

完成した処理系はBASIC Acceleratorと名付けインターネットを利用して公開している[6]．公開後，ほぼ1年経過しているが，今のところ，大きな不具合は見つかっていない．また，実行速度も期待通りに向上している．ただし，Full BASIC中核機能単位に含まれるTRACE文は未完成である．また，CHAIN文は配列に対応していない．その他，文字列変数の最大長宣言を無視するなど，細部の非互換がいくつかある．

本稿では，重要な問題に対して概略的な記述を行った．実際には，規格に合わせるためにより複雑な処理が必要になる．それらは，翻訳したできたPascalコードを見ることで確認できる．なお，処理系自体もソースコードとともに公開している．なお，翻訳してできたPascalプログラムはソース中にあるPascalユニットで定義されたルーチンを呼び出す構造になっている．

8.2. 最後に

数値計算による実験を行うためにCのように難しい言語を学ばせる必要はない．Full BASICで書かれたプログラムでも遜色ない速さで実行できるようになった．もっと多くの人に，数学や科学，工学を学ぶ過程でコンピュータを利用してもらいたいと願う．

【References】

1）「Windows環境におけるJIS Full BASICの実装」，白石和夫，情報処理学会論文誌：プログラミング Vol.41 No.SIG 9 (PRO 8)，pp. 52-61（平成12年11月）
2）http://hp.vector.co.jp/authors/VA008683/
3）「日本工業規格 JIS X 3003電子計算機プログラム言語 Full BASIC」，日本工業標準調査会，日本規格協会（平成5年11月）
4）Free Pascal - Advanced open source Pascal compiler for Pascal and Object Pascal - Home Page
http://www.freepascal.org/
5）Lazarus
http://www.lazarus.freepascal.org/
6）ダウンロード - Decimal BASIC Open Source Project - SourceForge.JP
http://sourceforge.jp/projects/decimalbasic/releases/?package_id=9955