RATFOR(合理化 Fortran)についての一考察

ISHIDA, Norimichi / 石田, 則道

(出版者 / Publisher)
法政大学工学部
(雑誌名 / Journal or Publication Title)
Bulletin of the Technical College of Hosei University / 法政大学工学部研究集報
(巻 / Volume)
22
(開始ページ / Start Page)
273
(終了ページ / End Page)
281
(発行年 / Year)
1986-03
(URL)
https://doi.org/10.15002/00004019

RATFOR（合理化 Fortran）についての一考察

石 田 則 道

Consideration of RATFOR

Norimichi ISHIDA*

Abstract

Ratfor is a preprocessor for Rational Fortran, which can be easily understood by any one familiar, with Fortran.

Ratfor are designed and developed at Bell Laboratory by Brian W. Kernighan, depending on "structured programming" with E. W. Dijkstra.

This paper describes behind of environment appear, principle of tool and future.

§1. はじめに

RATFOR は "Rational-Fortran" のことであり，構造化プログラミングを実現するための Fortran へのプリ・プロセッサである。これは E.W. Dijkstra が提唱した Structured Programming の考えを，Brain W. Kernighan によって設計され，ベル研究所で作成されたものである。

Ratfor の出現背景，この言語の文法と特徴を紹介し，Ratfor の今後について考察する。

§2. Fortran 言 語

今の計算機の形態をとる最初のものから，わずか40年の歴史しかないのに，その発達の様は社会の発展と相まって急激な変化である。その中で科学技術の計算機言語として Fortran は中心的な存在である。

計算機のハードウェアが出現し，それを稼動させる最も基本のものは機械語であり，これはとてもやっかいで何んら汎用性はなく，計算機毎に変わる命令である。誰でも使えるようにするにはもっと共通性があり，分かりやすくなければならない。このような背景の下で，アセンブラ言語が出現し，1950年代中ばには計算機の記憶能力を利用しコンパイラ（翻訳）と呼ばれる高級言語ができた。式の形で書けてそれを機械語に変換するもので，これが Fortran（*For*mula *Tran*slation）である。Fortran 言語を最初に手がけたのが IBM 社で，当時（1956年）の技術水準でこのような言語処理系を作り上げ，維持することは大変な作業であった。計算機を使いた

* 計算センター

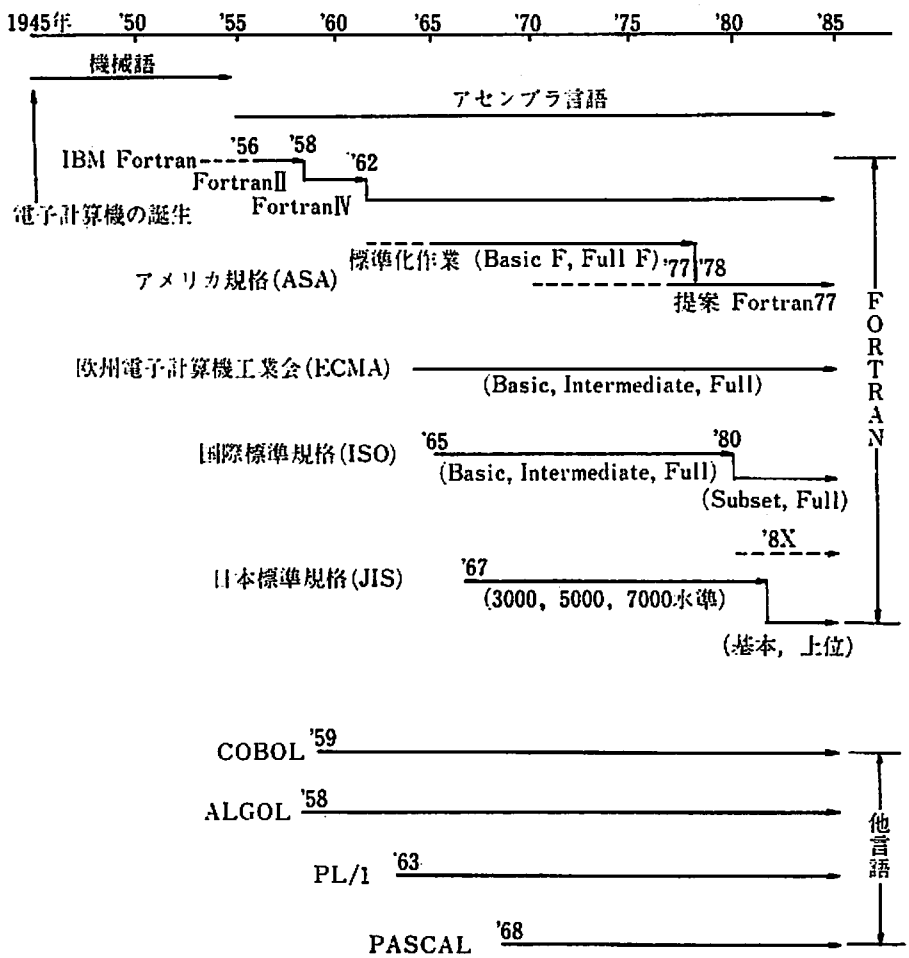


図1 Fortran を主にした高級言語の開発状況

い人が，一度，Fortran を覚えてそれが実用に絶える言語であったために，もう他の言語を覚えようとしなかった。いろいろな Fortran によるプログラムができてきて，除々にソフトウェアが蓄積されてくるし，それが次々と受け継がれ，Fortran のユーザは増加の一途をたどった。このように皆が Fortran を使い，その言葉で情報交換をしだすと他の計算機メーカも Fortran コンパイラを常備する必要がある。その頃は Fortran とは IBM 社の処理系をさすようになってしまった。他の計算機言語とともに Fortran も一つの言語として確立し，標準化され，規格化された(図1)。

とにかく，Fortran は先発の利を生かし多くの利用者はいるが，言語としてはいくつかの欠陥がある。最も欠けている点は制御構造とデータ表現であろう。

問題を解析し，その計算手順を具体的に表すことがプログラミングであり，文法にしたがって記述するが，その構造が全体として見通しのよいことが大切である。以前，プログラミングは一種の熟練工のような作業であり，計算が早いだとか，ステップ数が短かいとかがもてはやされた時もあった。その間，計算機の素子の方は次々と新しいものが作られ，ソフトウェアの立ち遅くれが目立った。プログラムの生産性，複数によるソフトウェアの作成と視点も変り，個人プレー

の時代は過去のものとなった。

それが1970年代に入ってソフトウェアの危機がさけばれ，E.W. Dijkstra によれば“Fortran などというものが存在していたことをわれわれが忘れ去る日が早ければ早いほどよい”とまで言わしめ，goto 文に依在するプログラムがいかに，よくないかを指適している。彼の発想と言われる構造化プログラミングが注目され，ソフトウェアに対する工学的アプローチが始まった。

§3. RATFOR について

Ratfor は Fortran 言語から goto 文と文番号を取り除いたもので，プログラミングの論理を明確にし，プログラムの読み書きをたやすくするプリ・プロセッサ言語である。

プリ・プロセッサとは前処理のことで，この場合 Ratfor で書いたソースプログラムを Fortran に翻訳する過程をいい，その翻訳結果（オブジェクト・プログラム）の Fortran は Fortran コンパイラの入力ファイルとして再び翻訳されるのである(図 2)。

3.1 RATFOR の文法

文法は図3示す。図中〔 〕は省略可能を意味する。以下に Ratfor の文を取り上げ，例を示しながら説明する。

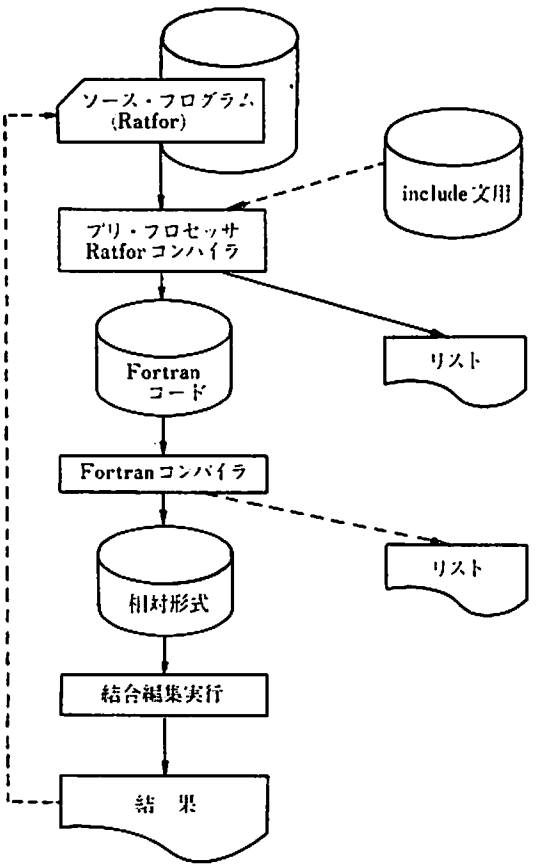


図2　Ratfor 処理過程

NAME　　Ratfor - Rational Fortran

DESCREPTION

statement grouping:

{statement 1 [; statement 2]}

decision-making:

if (condition) statement 1 [**else** statement 2]

loops:

while (condition) statement

for ([initialize]; [condition]; [increment])

statement

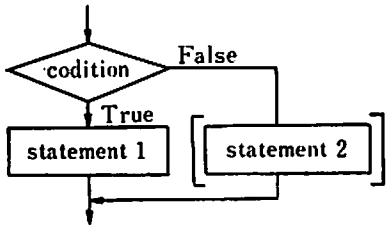
repeat statement [**until** (condition)]

do limits statement

break

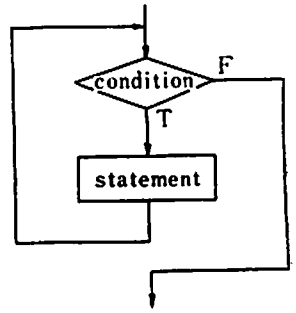
next

図3　Ratfor の文法



if (condition) statement 1
〔ELSE statement 2〕

1) **if 文** ある条件により2分岐するような制御構造は if 文によって処理するとよい。左のような流れの場合，Ratfor での if 文は下のように書くことができる。条件 (condition) を評価し，True のときは statement 1 を実行し，False のときは statement 2 を実行する。statement 1 及び 2 は Ratfor 文で書き else 句 (else およびそのあとの statement 2) が省略されたときは空となる。また，if 文はネストに書いてもよい。すなわち，statement 1 または 2 がまた if 文であってもよい。statement が複数の文の集まりからなるときは "{","}" で囲むと分かりやすい。



while (condition) statement

2) **while 文** 右図のような制御は while 文で表すことができる。条件の評価により True のとき，statement を実行し，再び条件に戻る。条件が False になるまで，このループは繰り返される。もし，最初の評価で False ならば statement は一度も実行されない。

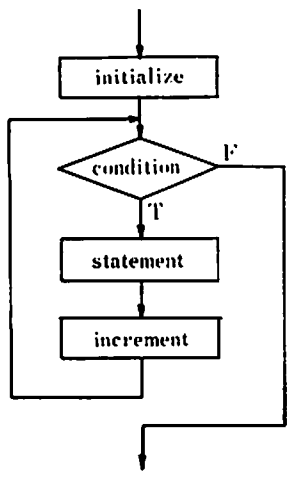
3) **for 文** 上述 while 文に初期値設定 (initialize) と増分 (in crement) の箇所が追加した形のものである。

まず，初期値設定し，条件の評価が True のときは statement を実行し，次に増分が行われ，条件の所に戻ってくる。この過程を False になるまで繰り返す。

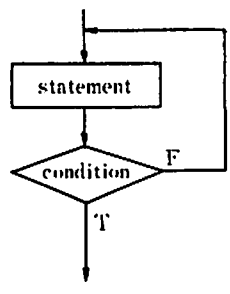
for 文の各部分は省略可能であるが ";" は省くことはできない。

4) **repeat 文** 流れ図のように，条件が statement の後で評価されるので，statement は必ず，1回は実行される。until 句を省略した場合は無限ループになる。

条件を最初に判定するのが while 文で，後で判定するのが repeat 文である。



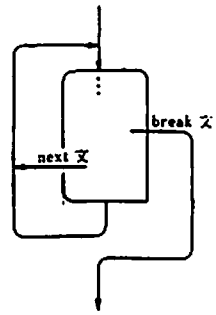
for ([initialize]; [condition];
[increment]) statement



repeat statement
[until (condition)]

5) **do** 文　Ratfor を使用する人は，だいたい Fortran プログラムの経験者と思われるが，Fortran の do 文から文番号を取り除いたものである。do 文の範囲が複数の文からなるときは "{" と "}" で囲めばよい。

do limits statement



6) **break** 文と **next** 文　Ratfor での while 文，for 文，repeat 文および do 文のループから抜け出すときは break 文（Fortran から Ratfor への変換規則参照）を使用する。next 文はループの残りの部分をスキップして，次のループに制御を移したいときに利用する。

7) **define** 文と **include** 文　define 文は Fortran のパラメータ文のように記号変数で定数を定義しておくことにより，プログラムの作成の自由度を増やすことができる。

例えば，

define (EOF, −1)

と定義しておくことで

while (getc(c)¬=EOF) CALL putc (c)

のように，プログラムが書ける。

include 文は "include file" の形で表現し，file の内容を Ratfor のソースプログラムの中に，組み込むときに使用する。各プログラム単位でよく出てくる文などは，事前に file に入れておくことにより，簡単に組み込むことができる。

次頁は，if 文，while 文，for 文，do 文を Ratfor で書いた例であり，それぞれ翻訳した Fortran とその実行結果を示す。

3.2 Ratfor の長所と短所

Ratfor は Fortran の欠点と言われる部分を補うものであり，Fortran で表現すると，とかく分かりにくくなりがちな概念を比較的簡単に書き表すことができる。これは Ratfor の論理構造が明確であると言うことであろう。

プログラムの読み書きが簡単であるということは大切であり，その点でも Ratfor は使いやすい言語である。例でも分かるように，それは次のようなことが考慮されていることに起因する。

1) 英小文字　小文字を使うことによって，大文字が基本の言語（Fortran, PL/1, BASIC など）よりプログラムが書きやすく，また読みやすい。

2) 自由形式　行のどのカラム（1〜72）から書いてもよく，";"（セミコロン）を区切りとして，一行に複数の文も書くことができるし，一つの文が複数の行にまたがることもできる。

3) 注釈文　"#"記号は注釈文を表し，どのカラムに書いてもよい。文のうしろに書くこと

```
$ type cif.r
#  check if-statement in Ratfor
 read *,a,b
 if (a<=b)
    {sw=0; print *,sw,a,b}
 else
    {sw=1; print *,sw,b,a}
 stop
 end

$ type cif.f
      READ *,A,B
      IF (.NOT.(A.LE.B))GOTO 23000
      SW=0
      PRINT *,SW,A,B
      GOTO 23001
23000 CONTINUE
      SW=1
      PRINT *,SW,B,A
23001 CONTINUE
      STOP
      END
$      run cif
23,12
  1.000000        12.00000        23.00000
FORTRAN STOP
$ run cif
23,125
 0.0000000E+00    23.00000        125.0000
FORTRAN STOP
```

```
$ type cif.r
#  check if-statement in Ratfor
 print *,"type-in a,b"
 while( a)=0 ) {
 read *,a,b
 if (a<=b)
    {sw=0; print *,sw,a,b}
 else
    {sw=1; print *,sw,b,a}
 break
}
 print *," ---end---"
 stop
 end
$     type cif.f
      PRINT *,11Htype-in a,b
23000 IF (.NOT.( A.GE.0 ))GOTO 23001
      READ *,A,B
      IF (.NOT.(A.LE.B))GOTO 23002
      SW=0
      PRINT *,SW,A,B
      GOTO 23003
23002 CONTINUE
      SW=1
      PRINT *,SW,B,A
23003 CONTINUE
      GOTO 23001
23001 CONTINUE
      PRINT *,10H ---end---
      STOP
      END
$ run cif
type-in a,b
4,23
 0.0000000E+00    4.000000        23.00000
```

```
$ type cfor.r
# Check for-statement
integer sum
n=10
for(i=1;i(=n;i=i+1)
  sum=sum+i
write(6,100) n,sum;100 format(" sum of 1 to ",I2,I5)
print *,
stop
end

$ typee cfortype cfor.f
      INTEGER SUM
      N=10
      I=1
23000 IF (.NOT.(I.LE.N))GOTO 23002
      SUM=SUM+I
23001 I=I+1
      GOTO 23000
23002 CONTINUE
      WRITE(6,100) N,SUM
100.  FORMAT(13H sum of 1 to ,I2,I5)
      PRINT *,STOP.
      END
$ run
_File: cfor
sum of 1 to 10    55
 0.0000000E+00
```

```
$ type cdo.r
# Check do-statement
 integer x(5),y(5),z(5),w(5,5)
 n=5
 do i=1,n; x(i)=i
 write(6,10) (x(j),j=1,n)
 do i=1,n {
    y(i)=i+1  # set to y-element
    z(i)=i+2
 }
 print *,"y and z"
 write(6,10) (y(j),j=1,n)
 write(6,10) (z(j),j=1,n)
 do i=1,n
   do j=1,n
     w(i,j)=i+j
 print *,"w"
 write(6,10) ((w(i,j),j=1,n),i=1,n)
 stop
10 format(5i3)
 end
$ type cdo.f
      INTEGER X(5),Y(5),Z(5),W(5,5)
      N=5
      DO 23000 I=1,N
      X(I)=I
23000 CONTINUE
23001 CONTINUE
      WRITE(6,10) (X(J),J=1,N)
      DO 23002 I=1,N
      Y(I)=I+1
      Z(I)=I+2
23002 CONTINUE
23003 CONTINUE
      PRINT *,7Hy and z
      WRITE(6,10) (Y(J),J=1,N)
      WRITE(6,10) (Z(J),J=1,N)
      DO 23004 I=1,N
      DO 23006 J=1,N
      W(I,J)=I+J
23006 CONTINUE
23007 CONTINUE
23004 CONTINUE
23005 CONTINUE
      PRINT *,1Hw
      WRITE(6,10) ((W(I,J),J=1,N),I=1,N)
      STOP
10    FORMAT(5I3)
      END
$ run cdo
  1  2  3  4  5
y and z
  2  3  4  5  6
  3  4  5  6  7
w
  2  3  4  5  6
  3  4  5  6  7
  4  5  6  7  8
  5  6  7  8  9
  6  7  8  9 10
FORTRAN STOP
```

Ratfor から Fortran への変換規則

文	Ratfor	Fortran	

if	if (condition)
	  statement 1
	else
	  statement 2
		
		L
		
		L1
			if (.not. (condition)) goto L
			  statement 1
			  goto L1
			continue
			  statement 2
			continue

while	while (conition)
	  statement
		L
		next
		break
		L1
			initialize
			if (.not. (condition)) goto L1
			  statement
			  goto L
			continue

for	for (initialize;
	  condition;
	  increment)
	  statement
		L
		n
		b
		L′
		L1
			initialize
			if (.not. (condition)) goto L1
			  statement
			  increment
			  goto L
			continue

repeat	repeat
	  statement
	  until (condition)
		L
		n
		b
		L′
		L1
			continue
			  statement
			if (.not. (condition)) goto L
			continue

do	do limits
	  statement
		
		n
		b
		L
		L1
			do L limits
			  statement
			continue
			continue

break	break		goto L1
next	next		goto L (while, repeat, do 文) goto L′ (for, repeat-until 文)

ここで L, L1, L′ はシステムが生成する文番号

ができる。

4) 文のグループ化 あるまとまった処理を複数の文で作るとき，"{" と "}" で囲むことにより，一つの文として扱うことができる。他言語の begin, end (ALGOL, PASCAL) や do, end (PL/1) に対応するが "{","}" の書き方の方が，分かりやすい。

5) 演算子 if, while, for のそれぞれの文，および until の condition において，演算子に以下のような記号を書くことも可能である。条件を見たとき，記号の方が直感的である。

Ratfor	Fortran	meaning
>	.GT.	Greater Than
>=	.GE.	Greater than or Equal to
<	.LT.	Less Than
<=	.LE.	Less than or Equal to
==	.EQ.	Equal to
¬=(又は!=)	.NQ.	Not Equal to
¬(!)	.NOT.	Not
&	.AND.	And
|	.OR.	Or

Ratfor がこのように，いくつか優れた機能がある反面短所もある。その一つは処理時間がかかるということであろう。Ratfor 自身がプリ・プロセッサであることを認識すれば，当然のことであるが，通常の Fortran での記述からの実行の前に，もう一つの処理が必要である。そしてプログラムの修正も Ratfor それ自身を変更する方がよいであろう。Ratfor の出力の Fortran プログラムでも修正可能だが，Ratfor から定形的に Fortran へ展開しているために，continue 文，goto 文を多く含み，読みずらく，分かりにくい。

また，Ratfor は特定の計算機システム (VAX コンピュータのオペレーテング・システム (OS) UNIX* での標準ソフトウェア) でしか使用できないということも問題である。

3.3 Ratfor の今後

現在，Fortran 言語の規格 (通称 Fortran 77) に，制御構造 (if-then-else) を書くことができるし以前のものに比べて改良されてきた。新しい規格 (8X) ではデータ表現 (動的機能，再帰的定義，配列処理)，プログラム書法 (自由形式) などが盛りこまれるようである。その一部はすでに Ratfor で使用できるのである。

Ratfor は特定の計算機システムの Tool であるがプログラムの作成が容易であり，それが抽象化として意味があり実用的である。このことはシステムを作る上で最も大切なことであり，多少処理に時間が費いやされても利用価値はある。今後も使われるであろう。

---

* UNIX とはベル研究所が開発した汎用の会話型 OS であり，ベル研の登録商標である。

§4. おわりに

いくつかの利点をそなえた Ratfor が，現在使える環境が乏しい。私も VAX が使用できたときの経験が下地になっている。その概念の一部を引用し，プリ・プロセッサの処理系を試作中である。ソフトウェア作りは大変労力と手間がかかる，その作業が少しでも緩和される Tool が必要であろう。

本研究は法政大学在外研究期間の一作業であったとを付記する。

参考文献

1) Brain W. Kernighan and P.J. Plauger: Software Tools, Addison-Wesley Inc. (1976).
2) Brain W. Kernighan and P.J. Plauger (木村泉訳): (プログラム書法), 共立出版。
3) 東京大学大型計算機センターニュース, Vol. 11, No. 6.
4) 石田晴久著: (UNIX), 共立出版。