ソフトウェア・ツールについて

若林信夫

1. はじめに

ソフトウェアシステムは通常，単発的に問題解決の手段となるのではなく，総合的に問題解決を支援する系である。ソフトウェアシステムは目的上，研究・開発用か，生産・商用に分類されるにせよ，対象別に，個人又は特定集団か，不特定多数を相手にすると分類されるにせよ，構成要素としてソフトウェアモジュールから組立てられている。ソフトウェアモジュールのうち，共通の目的に役立つことができるものを「ソフトウェア・ツール（ソフトウェアの道具）」と呼んでいる。例えば，テキストエディタやファイル処理ルーチンのような軽装の小道具や，良形構造をもった言語システムや，さらに大規模で管理可能なオペレーティングシステムは全て，このツールのひとつである。

近年，ソフトウェアシステムの拡大につれ，その生産性・保守性あるいは信頼性に対する認識が高まり，ソフトウェア作成の開発環境の整備が系統的に議論され，いくつかの成果が発表されてきた。従来，個人や企業のソフトウェアの秘密がベールに包まれていたが，次第にソースレベルで公開されるようになり道具箱に安心して良質の道具をおけるようになった。ひいては，改良が加えられたり新しい道具を開発させる刺激を誘発してきた。また，このことは，ソフトウェアシステムにいくつかの方法論的な反省を与えた。第一に，生産性・保守性向上をねらったプログラムの読み易さ，書き易さの向上である。それにはモジュラー性や明晰さを促進させた，できるだけ機種独立な高級言語で記述・交信されることが必要で，トリックプログラムや稚拙なプログラムが淘汰されることになる。第二に，実行時効率について慎重な配慮がなされる。しば

しば生産性向上と実行時効率は相反するといわれるが，全体的な生産性や保守性を損うことなく，満足水準の実行時効率は保証されねばならない。第三に，文書管理の完全性があげられる。どんなソフトウェアシステムも正確であるとか完全であることは保証されず，改版がなされる運命にある。改版に伴なう文書管理が十分でないとシステムは効率よく利用されない。

カーニハン・プローガー（以下，KPと略す）の「ソフトウェア・ツール」[7]は，ツール類を中心に，ソフトウェア作成の本質を明快かつ具体的に提示した好著のひとつである。(1) われわれは，同書を管理科学教育で採用し，実践と経験を積み上げてきた。本稿は同書を基礎に，上記三つの観点から，ソフトウェア・ツールについて検討する。

最初，ソフトウェア・ツールの基本問題を総論的に検討し，実現言語としての Ratfor プリプロセッサについて概説し，各論として，いくつかの使用中のソフトウェア・ツールについて述べる。付録では，Ratfor とその使用上の注意について述べる。

2. ソフトウェア・ツール総論

ソフトウェア開発にあたり，プログラマは十全にして注意深く，彼の環境を勘案し，仕様設計を行い，プログラミングにとりかかる。彼の環境とは使用計算機のオペレーティングシステム（狭義）やファイル中の応用プログラム，言語処理プログラム，ライブラリ，デバッギング支援，あるいは人的能力である。それらのうち，一般的な問題解決の支援になり，だれでも容易に使えるプログラム群をソフトウェア・ツールという。そのような目的に役立つソフトウェア・ツールには，プログラミング・ツールとドキュメンテーション・ツールがある。前者には，テキストエディタやクロスリファレンス（相互参照表），

原稿受領日　1980年1月16日

(1) ソフトウェアのライフサイクルには設計・記述・製造・評価・保守の段階があり各段階にソフトウェア・ツールが存在するとし，それらを解説したものに [13] がある。そのような細分化には若干無理が感じられるが，本稿はそのうち，記述と製造に重点がおかれている。「よい」プログラムについては [5]，[14] 参照のこと。

さらには，デバッガー，動的解析プログラム（DAP），マクロ処理系，トランスレータ，コンパイラ等が含まれる。後者には，テキストフォーマッタ，プリティプリンタ，流れ図作成ルーチン等が含まれる。伝統的にはこれらを指す用語として「ユーティリティプログラム」が使われてきたが，そこには広義のソフトウェア・ツールともいうべき，マクロ処理系や言語処理系は入らなかった。さらに，ユーティリティプログラムとソフトウェア・ツールの相違点として前者は全く外から与えられたもので内部の動きがよくわからないのに対し，後者は，ソースプログラム自体が公開されるので，内部が詳細に検討でき，文書化の自己記述性が保証される。

プログラマの環境整備の第一の要件はそのようなソフトウェア・ツールの整備にある。ツールは仕様設計の明確な，使い勝手のよい，信頼のおけるプログラム群でなければならない。ツールの需要が増せば，ツールは一層，高品質になるので，ツールの需要喚起も必要である。信頼のおけるツールの装備は結局のところ，ソフトウェア作成の生産性を高める。そのさい，会話的環境で，インクリメンタルに修正，実行していくことが生産性増加と切り離すことができない。ただし，ソフトウェア・ツールはしばしば会話的環境と組合せて議論されるがバッチ的環境をないがしろにすることはできない。

ここで「生産性」とは何であろうか，経済学的な考察をしてみよう。プログラミングの世界では生産性がソフトウェア開発にとって重要な因子と考えられ，若干の計量分析も行われているが（例えば，[10]）その測定には種々の疑問が残る。

プログラミングの生産性の計量分析では，月当り生産量（測定単位はソース・カード枚数）$=f$（人・月）のような定式化が行われ，最小二乗法で，対数型生産関数のパラメータを決定し，生産性を云々する。

しかし，ソフトウェア開発には創造的プログラミング，単純・定型的プログラミング，複写的プログラミングのように様々な側面があり，また，ツールの装備率も異なるので，アグリゲートして生産関数の推定をしてもあまり意味がない。第二に，プログラミングの生産性には，とくに定型的業務ではプログラ

マの経験による学習（learning by doing）の効果が大きい。独立変数として単純に人・月をとるのではなく，最低限効率性単位で測られた人・月がとられねばならない。さもないと，ブルクス［1］が批判したように，1人の人間と1カ月という時間は代替可能ではなく，納期に間に合わせるための人員増は品質と効率の低下をもたらす。他の独立変数の候補としては前記の装備率と人的能力があげられよう。第三に，従属変数として月当り生産量はカード枚数という点が批判される。カードによるバッチ処理でしかもマクロをもたないアセンブラといった前近代的な環境ならいざ知らず，昨今の環境では全く妥当しない。目的コードの大きさ，ソースの文字数，ＣＰＵ使用時間，等々の加重和とすべきであろう。

以上のことから「生産性」は日常かなり曖昧に使用されていることがわかる。また「生産性」とならんで「保守性」の重要さが強調されてきた。例えば，ゼルコビッツは，プログラミング活動のうち67％を保守に占められ，モジュールテスト８％，新規コーディング７％，統合化７％，設計５％，仕様，要求がそれぞれ３％であると述べているが，異論をはさむ人は少ない。

以上の考察から，通常のソフトウェア・ツールは高級プログラム言語の採用が必然的になる。ただし，全てを高級なプログラム言語で記述するのではなく，コミュニケーションとして使用されるべきことを意味する。例えば，言語Ｃははじめアセンブラで書かれ後にＣ自身で書かれた［8］ように当初から，高級言語の使用が不可能なこともある。逆にミニコン環境ではメモリ不足のために，中，大型機でのクロスコンパイラ方式をとったり，中，大型機で稼働している高級言語のコンパイラをハンド・アセンブリングにより，アセンブラ語で書くという開発途上段階を否定することはできない。

次に，ソフトウェア・ツールの記述のためにわれわれの環境[(2)]ではどんな高級プログラム言語を採用することができるか考察しよう。

（ｉ）Cobol

世界で最も広く使われている高級言語といわれるだけあって，純粋な数値計

(2)　MELCOM-COSMO 700（512KB，ディスク200 MB，OS は UTS/VS）を指す。

算が僅少のプログラムは一般的に実行速度が速く，目的プログラムの大きさも小さい。最近の Cobol 処理系には大型命令といわれるＳＥＴ命令，SEARCH 命令，文字処理命令（STRING, UNSTRING 命令）が備わっているので，ツールの記述言語としては有望である。しかし，それらの大型命令が欠けると，いたずらに行数が増え，機種依存のこともあり，プログラミングの保守・生産性を低下させる。ツールの中に含めるならば，ファイルの即時入出力ルーチンにとどめるべきであろう。

(ii) Fortran

Fortran は Cobol と対照的に数値計算を多く含み，入出力部分が比較的少ないソフトウェアでは圧倒的に速く，目的コードの最適化も行われる。また，どの計算機にも標準規格をもった処理系があるので移植が容易である。しかし，周知のように，再帰が許されないこと，データ構造が貧弱なこと，言語にあいまいさが残っていることのため，簡潔にして明解な，信頼のおけるプログラムを作るという見地からは劣る。

(iii) Pascal

Pascal は Cobol と Fortran の弱点をカバーし現代的な制御構造をもち，レコード構造はもちろん，ファイルやセットの機能もあり，ソフトウェア・ツールの記述のためには好まれる。生産性，保守性も良好である。しかし，現在のところ，ストリング処理の機能が劣る，入出力や実行速度が遅い，break, next 文がない，浮動小数点演算に誤差が混入する，コードの最適化ができないなど細い点でコンパイラに不満があり，ツールの本格的記述言語として今一歩のところにある。

(iv) BCPL

豊富な制御構造をもち，break, loop 文があるが本格的なコンパイラシステムを持たない限り，すべての点で Pascal よりも劣る。現に，本格的な BCPL コンパイラを持ち，多数のライブラリを備えている英国や西欧の大学では，ツールの開発はほとんど BCPL で行われている。文献［９］はＫＰ[7]を BCPL に自動変換し，効率よく利用している実際例である。

(v) PL/I

PL/I も広くソフトウェア・ツールの記述言語として使用されてきた。PL/I は他の高級言語よりも強力にビット，アドレス，構造体データ，ストリング処理，リスト処理が可能であるから企業側では好まれる。しかし，PL/I の機能が大きすぎるため，移植に問題があり，保守はしにくい。また実行効率も劣る。

(vi) SNOBOL 4

SNOBOL 4は文字列処理やリスト処理に向いているので，ソフトウェア・ツールの記述に適するはずであるが，構造的なプログラミングがしにくく，実行効率もさほどよくない。また，どの計算機にもあるわけでないので移植がしにくい。なお，ギンペル [3] は，SNOBOL 4 を用いてソフトウェア・ツールが非常に少数の行数で容易に作ることができることを豊富な例で示している。

以上，われわれが日常的に会話的に使用できる高級プログラム言語について，ソフトウェア・ツールの記述ツールの観点から検討してきたが，いずれも一長一短があり，決定的に優位なものはないことがわかる。記述者の得意な言語，サポートの十分な言語が選ばれている現状が首肯できる。

しかし，新たにメニューの追加が可能になった。すなわち，Ratfor プリプロセッサであり，次節で論じよう。

3. Ratfor の概要と移植

3·1 Ratfor 概説

Ratfor (Rational Fortranの省略。RatFor, RATFOR, ラットフォなどの書き方があるが Ratfor に統一する) は，1975年にB.W.カーニハン (Bell 研究所) により設計・製作された Fortran 用前処理言語である [6]。Ratfor の特徴を端的に述べれば，(1)現代的な制御構造 (文) がある。(2)マクロ代入ができる (define 文)，そして，(3)ファイルの算入が可能である (include 文) ことであり，Fortran の様々な短所を補っている。本言語は最初，DEC の PDP/11 の UNIX オペレーティングシステム上で開発され，多数のソフトウ

ェア・ツールが書かれた。それは教科書[7]に集録され，別売の磁気テープとともに Addison-Wesley 社から販売された。流通経路に乗った Ratfor は評判がよく，Fortran 系の前処理言語（例えば，Mortran, IFTRAN, Westran）の追随を許さず，ソフトウェア教育の場から本格的な生産の現場にまで流布していった。特に，磁気テープのソフトウェアはモジュラー形式で，構造的かつ物理的にも信頼性が高く，移植が容易であった。

テープを介して，Ratfor とそれによって書かれたソフトウェアが稼働すると，当然の帰結として，それらに反省点や改良点が湧き上がったが，その数はそれほど多くなく，かえって Ratfor の名声を高めるのに役立った。それらの批判は Ratfor に対する言語仕様，トランスレータとツールの実行効率，および使用の便宜さについての要望と改良案であった。まず，言語仕様に関しては

(1) Fortran に便利なコメントやホレリス文字，アンパサン付文番号が許されない。

(2) 現代的な制御構造[3]としての case 文がない。

(3) define 文や include 文が言語Cほど融通が利かない。

といった問題が提起されたが，それらはどれも教科書の演習問題的であって，改良案の実現は困難ではない（金田 [12] 参照）。

次に効率性に関しては，処理速度と記憶領域のオーバヘッドについて，いくつかの計測がなされ派生語が作られた。そのひとつは MOUSE[2] であるが，

(3) 制御構造は Pascal よりも BCPL に近い。Pascal ではループの途中から抜け出すには **goto** 文を使わないとすると，Boolean 変数の導入を必要とするが，BCPL は **break** や **loop**（Ratfor の **next**）で可能である。下に Ratfor と BCPL の対比を示す。

$(.....	$(.....	L1 : .....
..... **next**	..... LOOP	..... GOTO L2
..... ⇔	..... ⇔	.....
break	BREAK	GOTO L3
.....	.....	.....
$) **repeat**	$) REPEAT	L2 : GOTO L1
		L3 :
(Ratfor)	(BCPL)	

言語Cも Ratfor の延長にある。KP (p. 315) によれば，プリプロセッサの便利さに比べればトランスレータのコストはたとえ全体の2倍かかってもトリビアルであろうといっている。確かに保守性の重要さを考えると，読み易く，書き易い信頼性のある処理系は他の何ものにも換え難いだろう。Ratfor の場合，変換された Fortran の移植の容易さを考えると Ratforの効率のよい変換アルゴリズムや目的コードの生成は重要であり，できるだけ非効率さは避けなければならない。KP自身，1977年，記号表の表探索におけるベタサーチ（順探索）法はハッシュサーチ法に書き換えるべきだったと述べている（[2] の pp.39－40)。実際，C言語では表探索はハッシュサーチ法が使用されている。

最後に使用の便宜さについては，文字セットの機種依存が深刻である。例えば，小文字や，特殊文字がないために，Ratfor の効果が出ない点である。これは言語が悪いというよりハード的な使用環境が悪いことに帰因する。

Ratfor の構文，プログラミング上の注意，および使用法については付録で述べる。

3･2 Ratfor の移植

われわれは2種類の Ratfor テープを持った。最初は Addison-Wesley 社からの購入であり，次はアリゾナ大学の Icon グループからの購入であった。アリゾナ版は種々の改訂がなされ，使い勝手が良さそうなので，以下でいう Ratfor はアリゾナ版の使用経験から述べる。Ratfor で書かれた各種のツールもアリゾナ版ではかなりよく改良が加えられているが，アジソン版の全てを含んでいない。

どちらのテープも9トラック，800BPI，80文字長 (LRECL＝80)，800ブロック長 (10レコード/ブロック)，IBM 系 EBCDIC 文字セットで，われわれの計算機環境には少数の特殊文字を除き，適合した。テープ上には単一ファイルとして全部が書かれているが，それは多数のサブファイルから成っている（単一ファイル・マルチサブファイル)。それぞれのサブファイルの区別は先頭に見出し部分として，

#-h-　名前　大きさ　年 日 時　（アリゾナ版）

か，　　＝＝＝＝＝＝＝＝＝＝＝　　（アジソン版）

が書かれているかによりなされる。前者の形式はＫＰのアーカイブプログラム・ツールにより作られた。

前述のように，ソースプログラムの移植に伴なう困難な点は文字セットの不適合であるが，ＫＰの translit プログラム・ツールによりほとんど修復可能である（アセンブラなら TR, PL/I なら TRANSLATE 命令を使う）。ただし，特殊なグラフィック文字，制御文字は慎重に配慮されねばならない。逆スラッシュ（＼），垂直バー（｜），チルド（～），左中カッコ（{），右中カッコ（}）は同じ IBM 系でもコードが異なる。Ratfor では { と } はそれぞれ ¥（と¥）に ¬は＾又は～に，そして｜は！に代替される。

Ratfor 前処理言語の機種独立な版の処理系を作成する方法はいわゆる自力進行法（Bootstrapping）である。標準規格の Fortran で書かれたほぼ完全に[4]機種独立な Ratfor がある。この Fortran 版を翻訳，結合，実行を注意深く行なえば第０版の Ratfor が稼働する。

しかしながら提供されたこの Fortran 版は Ratfor で書かれた Ratfor をプリコンパイルしたもので冗長な Fortran プログラムであり，読み易くない。全体で2,100行あるが，ソフトウェア・ツールのエディタを利用すれば容易に読み易い Fortran が1,500行程度でできる。しかし，これらの作業は Ratfor の洞察，アルゴリズムの理解といったプログラミング教育上の効果はあっても大して効率が上がることは期待できない。すなわち，Ratfor 自身の洞察の方が効果がある。

自力進行法の考え方はプログラミングシステム作成の方法論に深い影響を

(4)　ここで「ほぼ」と断ったのは次の３つの例外があったからである。
(1)　サブルーチン名 exit があったが Fortran の第１次参照子としての予約語でもあるので exits と名前を換えた。
(2)　関数 GETCH (C, FD) の中に GOTO 10, GOTO 23023 が連続しておかれていたが，内容の検討の上後者の第１カラムにコメントＣを付けた。ただし，このルーチンは仮のもので利用者はアセンブラに書き換えることを要求されている。
(3)　ホレリス文字の中に小文字が含まれていたためすべて大文字にした。これらはサブルーチン error, symerr, および remark に現れた。

もたらした。つまり，Ratfor の前処理言語を誕生させただけで使用に十分耐えるものではないが，Ratfor で書かれているためにいくたの修正，改良を段階的に施していって幼年期から成長期に到達することができる。自力進行法は自分の核を次第に頑健にすることのプログラミングの方法論であり，われわれの Ratfor に対しても改版を試みている。

改版の際の最初の注意点は，入出力命令群のアセンブラ化である。Fortran の入出力はあまりに遅すぎる。とくに1文字入力の関数として GETCH(C, FD) と1文字出力のサブルーチンとしての PUTCH(C, FD) だけでもアセンブラ版を用意する必要がある（ここに FD はファイル記述子，C は対象変数）。これにより機種依存となるが，入出力の効率を犠牲にすることは Ratfor の生命でもある（但し，移植には，自力進行版を流布できる）。

改版の次の注意点は効率のよい算法の選択であり，これには前述のハッシュサーチ法をとる試みが考えられる。

また profiler (FORDAP, FORTUNE) を利用して効率を上げることも必要である。

一旦，Ratfor が動き出すと，Ratfor の第0版からでも Ratfor で書かれたＫＰのソフトウェア・ツールが翻訳・実行できる。種々のツールの感触と醍醐味を味わううちに自力進行法の利点を知り，さらに改良が加えられる。Ratfor はツールの記述にとって多数の便益と多数の費用をもった前処理言語である。

次節ではそれらのツールの主なものを紹介しよう。

4. ソフトウェア・ツール各論

Ratfor が利用可能になると同時に，ソフトウェア・ツールが急速に豊富になった。メーカーの提供するユーティリティは便利であっても小回りが利かず，今まで一回だけ使われるプログラム（・ツール）をいくつ作成したことであろうか。

実際，良いハードウェアと良いコンパイラが提供されているだけでは良いプログラム環境にあるとはいえず，良いソフトウェアを作成することはできない。もちろん，良い道具が揃っていても良いプログラムが作れるとは限らない

し，良い道具も有機的，系統的にまとまっていなかったり，良い道具に良いドキュメンテーションが付いていないと効果は薄い。

以下，Ratfor で書かれたＫＰからの種々のツールやブロック構造言語に適したツールのうち，日常，有用と思っているものを紹介しよう。

(1) エディタ　　ユーティリティの行エディタはテレタイプ用にできているため頻繁に改行し，操作性もブロック言語には不向きである。ＫＰの文脈エディタは多様性があり，ツールの中の「王様」である。エラー回復や人間・機械心理も考慮されていて操作性はよい。難点は実行速度がやや遅いこと，CRT ディスプレイ（80字×24行の画面）によるカーソル編集ができないことである。

(2) ソースプログラム中の文字数，行数，単語数の計数プログラム　　自分のプログラムを他者のそれと比較するさい，文字数，行数，単語数を比較することにより優劣を論じることができる。ここで行は改行でおわるもの，単語は「白ブランク」（空白，改行，タブコード）に囲まれたものをいう。

(3) ファイル比較プログラム　　ソフトウェア開発途上でしばしば自分のファイル中に類似のプログラムをもつことがある。それらはどのように異なっているかを指摘してくれるプログラムである（ＫＰの練習問題3－4，p. 71)。Pascal まわりのファイル比較プログラムもある（*"Pascal News"* #12と#15の Software Tools 欄参照)。

(4) 相互参照表付きソースプログラムの節約印刷　　相互参照表（クロスリファレンス）は言語のオプションを指定することにより得られる場合もあるが１パスの処理系では，ツールとして別に用意するほかない。ＫＰの練習問題4－35として登場するが極めて重要なツールである。文献 [11] と [14] には各種の相互参照表プログラム例がある。手続き（ルーチン）名と一般の変数とを区別したり，各出現回数を出力すると，便利なデバッグ支援となり，移植されたプログラムを読むのに不可欠である。

ソースリストの出力も１ページを左右に使うと用紙の節約になる上，読み易さも増す。但し，一行に納まらないとき，適当な折り曲げが必要である。

(5) DAP（動的解析プログラム）　　D.E.クヌースの提案した profiler

はプログラム中に現れるすべての実行文の実行回数を計数するプログラムである。わが国では九大牛島グループにより種々の言語のDAP(すなわち,Fortran, Cobol, Pascal, Snobol および Ratfor) [15] が作られている。入出力部分が大きくないプログラムの実行状況を正確に把握することができ，全体の実行効率を改善できる。また，ミスパンチの発見にも役立つツールである。

(6) テキスト・フォーマッタ　　適当なプリンタ上にソースや文書をきれいに出力するプログラムで，いわば文書作りのツールである。14種の異なるコマンドに応じてテキストフォーマットの機能を実行する。例えば，標題に下線を引いたり，段付け (indent) を行ったり，文字の置換え (translit) をすることができる。

(7) マクロ処理系　　Ratfor にはマクロ代入を実行するごく単純な機能しか含まれていないが，KPのマクロ処理系は引数つきマクロ，条件つきマクロ，組込みマクロが可能なマクロ処理系である。マクロ機能のおかげで Ratfor の自力進行の改版も含め，大きなソフトウェアの移植が容易になった。

(8) Ratfor トランスレータ　　Ratfor トランスレータは Ratfor 自身でテープに記述されているので，自力進行が容易にでき，より効率の良い版をもつことができる。

(9) Icon 言語処理系　　Icon はその源流が SNOBOL 4 と SL 5 にあるようにストリング処理用の言語で，Griswold, Hanson らにより設計され，Ratfor を用いて作成されている[4]。

(10) その他　　KWIC (入れ換え索引)，整列，アーカイブ，FIND, COMPRESS, COPY, structurer (Fortran・Ratfor 変換ルーチン) 等々。

以上は，われわれの環境の下で Ratfor を中心に移植されたソフトウェア・ツール群である。現在われわれは，詳しい文書や使用法を整備中である。ツール (道具) は使われなければ何の価値もない。

5. おわりに

1970年代はハードウェアの低廉化に伴なって計算機能力が爆発的に進歩し，

日常の活動に計算機がより身近なものになった時代であった。ソフトウェアないしそれを記述するプログラム言語も雨後の筍のように出現し，一日に十個の言語を操ることすらあった。にも拘らず，日常の研究調査の活動が飛躍的又は比例的に進行したとはいえない。豊富さの中の貧困の一因はソフトウェア・ツールの貧困さに求められよう。あるいは，総合的なソフトウェアシステムの欠陥や不足にあるといってもよい。1980年代は愈々，超 LSI の低廉化が進行し，ソフトウェア危機がより深刻になるかもしれない。米国国防総省(DOD)の承認した Ada やバッカス(J. Backus)の関数的プログラミングは抜本的な対策となるだろうか。筆者は，本稿で検討してきたようなソフトウェア・ツールがもっと公開され，総合的なソフトウェアシステムが確立されることが重要であると考える。いずれにせよ，1980年代がソフトウェア研究にとってどんな時代となるかは計算機をめぐる政治経済・科学技術の発展と無関係ではない。

最後に，1970年代はプログラミングが知的な生産活動とみなされ，それを検討し，一層高い水準を達成せしめる実践と経験の教育の重要性が広く認識された時代であった。またそれは今後とも継続されるであろう。

付録 Ratfor の構文と使用法

[1] Ratfor の構文

Ratfor の構文は BNF 流に書くと下記のようになる。

<プログラム>::=<文> | <プログラム><文>

<文>::=**if** (<条件>) <文> | **if** (<条件>) <文>**else**<文>
| **while** (<条件>) <文>
| **for** (<初期状態>；<条件>；<更新>) <文>
| **repeat**<文> | **repeat**<文>**until** (<条件>)
| **do**<範囲><文> | <数字><文>
| **break** | **next**
| **define**(,) | **include**<数字>
| {<プログラム>}

| <その他>

<その他>::=Fortran のIF文，DO文，第1カラムのC，第6カラムの継続符号を除く Fortran 処理系の許す<文>

<数　字>::=<23,000台を除く10進数>

<範　囲>::=Fortran のＤＯ文の<範囲>

[2] Ratforプログラミング上の諸注意

上記の Ratfor の構文は以下の意味論に従って解釈される。

(1) **if** (<条件>) <文>，**if** (<条件>) <文>**else**<文>，**while** (<条件>) <文>，**repeat**<文>，**repeat**<文>**until** (<条件>) は Pascal や BCPL と同じ使い方である。<条件>を ( ) でくくる点 注意が必要である。

(2) **do**<範囲> <文> は Fortran の DO 文から文番号をとったものである。<文>は複合文のとき¥(と¥) でくくる。

(3) **for** (<初期状態>；<条件>；<更新>)<文>は次の等価な Fortran 文に置き換わる。

```
          <初期状態>
n1        IF (.NOT. (<条件>)) GO TO n2
          <文>
          <更新>
          GO TO n1
n2        CONTINUE
```

(4) <数字> <文>は <文> の前あるいは行の先頭に現れる数字は文番号とみなし1～5カラムに入り，7カラムめから<文>が続く。

(5) **break** は，**while**，**for**，**repeat**，**do** 文のループの中からとびだすときに用いる。**next** はその回のループの残りの処理は行わず，次の回のループ処理を行なう。

(6) **define** の書式は **define** (文字ストリング1，文字ストリング2) で，この文以降の全ての文字ストリング1を文字ストリング2に置き換えて処理する。再定義されない限り，ソースプログラムの最後まで有効である。

(7) **include** の書式は **include** 論理機番　　で論理機番に割り当てられたファイルの内容を Ratfor プログラムの一部としてこの文の場所に算入する。算入されたファイルの内容は評価され，その中に他の **include** が4つ以内ならあってもよい。JCL の ASSIGN 文の追加はいうまでもない。

(8) 1～72カラム内であれば自由形式でソースプログラムを入力することができる。次行への継続の場合，＜文＞のときには下線記号（_）を用いる以外文脈に依存して無原則に継続する。

(9) 1行に複数文を書くことができる。文と文の間は；(セミコロン)で区切る。この場合は継続行は許されない。

(10) コメント（注釈）は'#'を先頭におき，文の途中でもよい。

(11) 文字定数はすべてクォート（'）かダブルクォート（"）で囲み，1行以内に納める。もし'を含むときは"で囲み，"のときは'で囲む。

(12) 比較演算子，論理演算子

＞，＞＝，＜，＜＝，＝＝，は，＝，&，｜，は，それぞれ.GT.，.GE.，.LT.，.LE.，.EQ.，.NE.，.AND.，.OR.，.NOT. に対応し，いずれをも使用できる。また {と} はそれぞれ，¥(と¥)，￢は～又は～，｜は！に代替される。

[3] Ratfor前処理言語の使用法

小樽商科大学の計算センター (MELCOM－COSMO700) では次のようなJCL（ジョブ制御言語）を用いて利用することができる。なお，バッチ処理，TSS（端末）処理いずれも可能であるが，以下の説明はTSS（端末）処理に限る。

```
! SET  F : 5/TEST1      (ここで TEST1 は Ratfor ソースプログラムの入ったファイル名)
! SET  F : 6 ME         (ソースプログラム上のエラー出力は自分のところ)
! SET  F : 7/TEMP ; OUT ; SAVE   (TEMP は Ratfor ソースがFortran プログラムに変換されたファイル名)
! RATFOR. P022103       (ロードモジュールがアカウントP022103にある)
```

これにより通常（エラーがない場合），

EXECUTION F10

＊STOP RATFOR→FORTRAN

が出る。そのあと

!FORT　TEMP

とやると

EXT. FORTRAN IV, VERSION F10

OPTIONS>

と出るので Fortran のコンパイラオプション（例えば，ＮＳ，ＢＣとかＮＳ，ＬＳ，ＢＣ）を選択入力する。これにより，コンパイルされて，ロードモジュールになっているので実行させる。データを読む必要があったり，結果をその場で見たければ

!SET　F:5　ME

!SET　F:6　ME

!RUN

とすれば Ratfor のソースが実行される。

!RUN のあと，割付けサマリ等が出力されるがわずらわしい場合にはそれ以前に!SET M:LL NO を入れるとよいが，リンケージのエラーまで出力されなくなるから注意が必要である。また，大きなプログラムや特定のライブラリを必要とする場合には，!RUN の前に!LYNX ¥,(.ライブラリ名)が必要なことはいうまでもない。**include** 文を必要とするときも，附加的な!SETが必要である。

引 用 文 献

[1] F. P. Brooks, *The Mythical Man-Month,* An Essay on Software Engineering, Addison-Wesley, 1974.

[2] D. Comer, "MOUSE 4: An Improved Implementation of the RATFOR Preprocessor," *Software-Practice and Experience,* Vol. 8, 35－40 (1978).

[3] J. F. Gimpel, *Algorithms in SNOBOL4,* Wiley and Sons, New York, 1976.

[4] R. E. Griswold, D. R. Hanson, and J. T. Korb, "The Icon Programming Language: An Overview," *SIGPLAN Notices,* Vol. 14, 18－31 (1979).

[5] B. W. Kernighan and P. J. Plauger, *The Elememts of Programming Style,* McGraw-Hill, 1974. (木村泉訳『プログラム書法』 共立出版, 1975)

[6] B. W. Kernighan, "RATFOR-A Preprocessor for Rational FORTRAN," *Software-Practice and Experience,* Vol. 5, 395－406 (1975).

[7] B. W. Kernighan and P. J. Plauger, *Software Tools,* Addison-Wesley, 1976.

[8] B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, Inc., 1978.

[9] C. R. Snow, "The Software Tools Project," *Software-Practice and Experience,* Vol. 8, 585－599 (1978).

[10] C. E. Walston and C. P. Felix, "A Method of Programming Measurement and Estimation," *IBM System Journal,* Vol.16, No. 1 (1977).

[11] N. Wirth, *A Collection of Pascal Programs,* ETH Technical Report No. 33, 1979.

[12] 金田康正,「RATFOR の改良」(記号処理研究会資料6－1, 1978, 11, 東大大型計算機センター, センターニュース, 11－1, 1979).

[13] 国井利泰他,「小特集, ソフトウェア・ツール (Ⅰ), (Ⅱ)」情報処理, 1979年6月号, 8月号.

[14] P. S. 委員会,『よいプログラムを作るにはシンポジウム報告集』情報処理学会プログラミングシンポジウム委員会, 1979.

[15] 牛島和夫『Fortran プログラミングツール』産業図書, 1979.