解　説

FORTRAN をめぐるソフトウェア工学的ツール*

石　田　晴　久**

1. はじめに

近年，構造化プログラミング[1),2),3)]やソフトウェア工学をめぐる研究や論議が盛んになるにつれて，プログラミングの（とくにチームによる）やり方やプログラミング言語の仕様について幅広い注意が払われるようになったのは喜ばしいことである．PASCAL[14)]を始めとして，構造化プログラミング向きの新しい言語もいくつか設計され使用され始めている．

しかし，世の中の多くのコンピュータでは，とくに多くの科学技術者によって，相変らず最も広く使われているのは FORTRAN 言語である．**表-1** の数字は非常に極端な例といえるかもしれないが，東京大学大型計算機センターのように圧倒的に FORTRAN ユーザの多い計算センターは決して珍らしくない．このような現状をみるとき，少くとも FORTRAN ユーザにとっては，ソフトウェア工学の進歩は単独プログラマによる FORTRAN プログラミングにとっても利益をもたらしてくれるものでない限り，余り意味がないといわなければならない．そこで本稿で考えてみたいのは，最近のソフトウェア工学的なコンピュータ技術の進歩が FORTRAN プログラミングにどのような影響を与え，どのようなプログラミング・ツール[5)]（道具）をもたらしつつあるかということである．

表-1 東京大学大型計算機センターにおける各種言語の利用状況（1976 年 2 月のジョブ 71,873 件について）

	コンパイラ使用ジョブの割合	相対比重（全体を 100% として）
FORTRAN	74 %	97 %
PL/I	0.8%	1.1%
アセンブラ	0.8%	1.1%
COBOL	0.2%	0.2%
BASIC	0.1%	0.1%

したがって本論は一種の FORTRAN 擁護論といえなくもない．もちろん FORTRAN は最近のソフトウェア工学の見地からすれば排斥されるべき悪い言語の代表例という見方もできるが，そうした批判にもかかわらず FORTRAN が依然として広く使われる理由は次のように整理できるであろう．

（1） FORTRAN は立派に役に立つ言語である．

（2） FORTRAN は中級ミニコン以上のほとんどすべてのコンピュータで利用可能である．「FORTRAN は雑草のごとく，どのコンピュータにも生えている」のは事実である．

（3） FORTRAN コンパイラは非常によいものが数多く作られており，オブジェクト・プログラムの実行効率も実用上十分高い．

（4） 各種サブルーチンや応用プログラム・パッケージのぼう大な蓄積がある．したがって他の言語にとっても FORTRAN とリンクをとる必要のあることが多い．

（5） 上記(2)の高い可用性を反映して，JIS や ANSI などの標準規格も制定されており，FORTRAN プログラムの互換性はかなり高く，システム間でのプログラムの流通に便利になっている．

（6） FORTRAN については数多くの教科書や参考書が出版されており，学校でもよく教えられていて，学習する上でもとりつきやすい．

いずれにせよ，以上のような点はとくにあげてみるまでもなく，FORTRAN は非常に強力な言語である．歴史的要因からこのように強力になったプログラム言語が将来使われなくなるということは非常に考えにくい．そこでこの前提に立てば，将来ともソフトウェア工学および一般のコンピュータ技術の進歩を単独プログラマによることの多い FORTRAN プログラミングに反映させる努力を続けることは非常に重要であろう．

* FORTRAN-related software-engineering tools by Haruhisa ISHIDA (Computer Centre, University of Tokyo).

** 東京大学大型計算機センター

2. FORTRAN のプログラミング・スタイル論

FORTRAN のプログラミング・スタイルに関して昔からよくいわれてきた教訓の代表例は，プログラムの構造（DO のネスティングのレベルなど）がよく分るように，各文のはじめに適当に空白を入れよ，という種類のものであった．これはプログラムをカードで入力する場合には確かに悪くない教えであるが，自由形式で入力することが許される（ユーザにとって便利な）TSS 用の FORTRAN 処理系を使う場合にはこれを守るのは事実上不可能である．後述のように，この種のスタイリングは今後は，コンパイラのオプションにでもして，機械にまかせるべきであろう．

最近の構造化プログラミング論の FORTRAN への影響は，B. W. Kernighan らの "Elements of programming style"（プログラム書法）によく現われている[6]．この本は，各種の教科書や参考書にみられる FORTRAN および PL/I の悪いプログラムの例をあげ，それをもっとすっきりした誤りのないプログラムに改良するにはどういう考え方でどうすればよいか，を具体的に示したものである．この本であげているよいスタイルを導くためのガイドラインは多くの人が多かれ少なかれすでに守っているものと思われるが，たとえば次のようなものがある．

（1） GO TO 文はなるべく使うな．
（2） サブルーチンを多く使って，プログラムをモジュラーに構成せよ．大きなプログラムは小さく分けてテストせよ．
（3） 算術 IF 文はなるべく使うな．論理 IF 文を使え．
（4） 単純明確で直接的な表現を使い，トリック（名人芸）は使うな．
（5） くり返しをさけるのに配列を使え．プログラムを単純にするようなデータ構造を使え．
（6） 分りやすく混同の恐れのない変数名を使え．

スタイル論の観点からみたとき，FORTRAN で非常に困るのは，ブロック構造がないことである．これを補うため，筆者はよく

```
CALL FIND (THIS, THAT)
IF (THIS.EQ.THAT) CALL DOTHAT
IF (THIS.EQ.THAT) GO TO 100
```

のように同じ判定式をもつ IF 文を繰り返したり，

```
IF (THIS.EQ.BOY) CALL BOYS
IF (THIS.EQ.GIRL) CALL GIRL
```

といった構文を使うが，これは上述のガイドラインに沿ったスタイルの例といえよう．

こうしたスタイル論の立場からみると，現在の教科書や参考書には悪いスタイルのプログラムがかなりみられることは確かである．今後はすっきりしたスタイルのプログラム例を多くのせた参考書が望まれる．理想的には「シェークスピアの FORTRAN」，「志賀直哉の FORTRAN」とでも呼べるような FORTRAN の名文集があって，プログラマがそれらをお手本にできるようになるとよい．

次にスタイル論で今後さらに考慮すべきことは，バッファー・メモリや仮想記憶などのメモリ・ハイアラーキーやアレイ・プロセッサに対する考慮である[7),8)]．この点からは FORTRAN には限らないが，プログラミングの書き方にはワーキング・セットを小さくするために，たとえば次のような要請が出てくる．

（1） モジュール（entry）は機能の類似ではなく，使用時の関係の近さでまとめる．
（2） めったに生じない例外事項の処理は別のモジュールで行う．
（3） データはメモリ内の格納順にアクセスする．

```
S=S+A(j, i)*B(j, k)
```

はよいが，次の 2 例はよくない．

```
S=S+A(i, j)*B(j, k)
A(k, j)=B(j, k)
```

（4） 読み出すだけのデータと読み出し書き込みを行うデータとは領域を分ける．
（5） リスト構造はなるべく使わない．
（6） 入出力における DO 型並び〔(A(I), I=1, 10) など〕はもとのプログラムにその都度戻るため能率が悪い．
（7） データはプログラム内部に置かず，同時に使うものはまとめて COMMON に入れる．
（8） データは作るそばから使う．まとめて作ってまとめて使うのはよくない．

以上のような注意のうち，とくに(1)，(5)，(7)あたりは構造化プログラミング的な注意とは矛盾する面もある．やはりメモリ・ハイアラーキーに対する考慮はプログラマの心掛け程度にして，本当の効率化は後述のソースおよびオブジェクトの自動最適化にまかせるのが本筋であろう．

3. ソース・プログラムの扱い

FORTRAN はソース・プログラムの入力媒体とし

てもともとはカード（固定形式，文は 7-72 カラム）を想定して設計された言語である．カードを使う限りはいわゆるテキスト・エディタのようなオンラインの編集機能は必ずしも必要ない．しかし近年大きなディスク・ファイルが使えるようになり，TSS あるいはミニコンの対話的利用が増えるにつれて，FORTRAN を自由形式で入力して，テキスト・エディタで修正することが盛んに行われるようになってきた．これはとくに数行を修正してはコンパイルしてランさせることを繰り返す（これは開発法としてはよくないといわれそうだが）場合には非常に便利である．

しかし便利に使えるためには，ソース・ファイルに対するすぐれたエディタがなければならない．そうした強力なエディタの1例は，MIT，ダートマス大学，ベル研究所などで使われている QED である．たとえば，プログラムを十分モジュラーにして，各ルーチンの頭に C－－－で始まるコメントをつけておけば，QED で

g/SUBROUTINE|ENTRY|FUNCTION|C－－－/p

という「テキスト全体 (global) の中で，SUBROUTINE あるいは ENTRY あるいは FUNCTION あるいは C－－－を含む行（それらで始まる行という指定も可）があれば，それらの行をすべて印字せよ」という意味のコマンドを出すことにより，プログラムのドキュメンテーションをうることも可能である．

次にドキュメンテーションの観点からは，前にちょっと述べたインデンテーションの機能を含めた FORTRAN の文書化のためのソフトウェアも面白い．この種のプログラムは beautifier, pretty-printer[10], POLISH などさまざまな名前のものが開発されているが，たとえば文番号を整理してつけかえる機能などは，プログラマ個人の趣味には合わないこともありうる．しかし将来は FORTRAN コンパイラのオプションとして，ソース・プログラムを美化して印字する機能はあってもよいであろう．

同種のプログラムには各種のコンバーターもある．変換のモードとしては，文字コード設定（6ビット BCD, EBCDIC, EBCDIK, ASCII, JIS，カード・コードなど），大文字・小文字変換，ソース・ファイル内の行番号のつけかえ，行番号の移動（行の先頭からカラム 73～80 へなど），自由形式／カード形式変換などがある．こうしたコンバーターは今後コンピュータ・ネットワークが発達して，異種のコンピュータ同志が交信するようになると非常に重要になる．たとえば大文字・小文字変換は MIT の MULTICS やベル研究所で開発された UNIX (PDP 11/45) のように小文字ベースのシステムを相手にするときに必要になる．なおコンバータおよび構文チェックの機能はテキスト・エディタに含ませることも可能である．

ソース・プログラムを対象とするソフトウェアとしてはソースから流れ図を作るための AUTOFLOW といったものも使われている．これも文書化を助ける一手段であるが，トップダウン構成でモジュール化を徹底すれば流れ図は不要とする構造化プログラミングの立場からすると，将来とも使うべきソフトウェアとはいえないかもしれない．

ついでにいえば，みんながプログラムをソース・ファイルに入れるようになると，問題になるのは，ファイル・システムの容量である．ファイルの容量不足という事態は当然起こりうる．これを救うひとつの手段は，余りひんぱんには使わないファイルを自動的あるいは半自動的に磁気テープに移す archive（古文書保存）の機能である．これは MIT の MULTICS などで使われており，archive したファイルのリスト（カタログ）はオンラインで保持され，ユーザが取り出しを要求したファイルは数時間内には磁気テープからディスクに移されるようになっている．したがってユーザは自分で磁気テープをいじらなくてよい点がミソである．この機能は将来，磁気テープ自動倉庫に相当する MSS（マス・ストレージ・システム，IBM 3850 型など）が普及すれば比較的簡単に実現されることを注意しておきたい．

4. オブジェクト・ファイルの扱い

ファイルの使える FORTRAN 処理系ではオブジェクト・プログラムをファイルに格納することもよく行われる．そうしたオブジェクト・ファイルの編集が問題になるのは，数多くのサブルーチンのうちのごく一部だけを修正して，コンパイルさせ，さらに実行させることを繰り返す場合である．このとき修正していないサブルーチンまで改めてコンパイルし直すのは明らかに効率が悪く，そうかといってルーチンごとに別々のソース・ファイルに入れるのも面倒な話である．

一部のルーチンだけをコンパイルして，オブジェクト・ファイルのその部分だけをさしかえるのは，東京大学大型計算機センターの HITAC 8800/8700 の OS 7 では，ソース・ファイルの中がリジョン（ルーチンに対応可）に分けられることを利用して，リジョンの

さしかえという形で行っている．しかしこの場合は修正すべきリジョンの名前はいちいち指定しなければならない．

ユーザからみて，恐らくもっと便利なのは，ソース・ファイルは全体としてまとめてテキスト・エディタで扱い，再コンパイルのときには，修正のあったルーチンのみを自動的にコンパイルし直すことであろう．しかしこのためには，ソース・プログラム中のどのルーチンの中が修正されたかを，エディタが何らかの形で記録し，それをコンパイラあるいはオブジェクトのファイル・システムあるいはリンケージ・エディタに伝える機構が必要となる．こうしたことを行うのに，しいてFORTRAN専用エディタを設けるのがよいかどうかは問題であるが，とにかくソース・プログラムをちょっと直してすぐランさせてみるということが簡単に行える機構はユーザにとっては非常にありがたい．

なおオブジェクト・プログラムをファイルに入れる場合，リンク前のモジュール独立のいわばルースな形で格納するか，あるいはリンク後の圧縮されたコンパクトな形で格納するかの選択がある．後者の場合，ファイル・スペースは少くてすみ，再実行の際には速いが，前述のような意味でのモジュール単位のさし変えは通常不可能である．しかしシステムによっては，前のarchive機能の一部として，各オブジェクト・モジュールの名前やサイズや作成日時を示すインデックスを用意した上で，いくつかのモジュールをまとめて圧縮した形でarchiveファイルに格納し，後でそのインデックスを頼りにモジュール単位のさしかえを可能にしている例もある．

5. FORTRANのプレコンパイラおよびジェネレータ

FORTRANの言語仕様は初期のものに比べれば今日のJIS 7000レベルでもかなり大きくなっている．しかも大抵の大型機のFORTRANはJIS 7000を上回ることをうたい文句にしている位で，さらに拡大された言語仕様をもつに至っている．このまま拡大が続くと，10年後20年後にはFORTRANがPL/I位に大きくなってしまわないとも限らない．もちろんこれではFORTRANのよさは失われるとともに，拡大された仕様は結局少数のユーザにしか使われないから，言語仕様の拡大は決して望ましいことではない．

しかし反面，コンピュータ技術の進歩とともに，特殊な要求をもつユーザから言語仕様の拡大（アメリカで検討されているFORTREVなど）は当然要求される．この要求に答えるひとつの手としては，FORTRANの言語仕様はそのままにして，その代りに拡大された言語仕様で書かれたFORTRAN風のプログラムを標準FORTRANに変換するプレコンパイラ（プレプロセッサ）を用意することが行われる．そうしたプレコンパイラは小型機で標準規格を満足するコンパイラがないときに，基本FORTRANに変換するための手段として使われる例もある．

また最近は構造化プログラミング論の影響を受けて，FORTRANにブロック構造や自由形式などを導入した新しい言語仕様を設定し，それで書かれたプログラムを標準FORTRANに変換するためのプレコンパイラを作る試みが内外で盛んに行われている[11]．KernighanのRATFOR (Rational FORTRAN) はその1例である[12]．これはUNIXの主力言語Cに仕様を似せたもので，{ }の記号でブロックを定義し，DO…WHILE, REPEAT…UNTIL, IF…THEN…ELSE, CASEなどの構文が使えるようになっている．〔教育用として有名なWATFIV（第1版レベル4）にもIF-THEN-ELSE, WHILE-DO, DO CASE, EXECUTE, REMOTE BLOCK, WHILE-EXECUTE, AT END DOなどの文が導入されている．〕

FORTRAN擁護論の立場から，さらにすすめたいのは，これらのプレコンパイラそのものを標準FORTRANあるいは基本FORTRANで書くことである．こうしておけば，新しく追加された言語仕様はプレコンパイラそのものと一緒に他システムに移すことが可能となり，互換性が保てることになる．前述のRATFORのコンパイラはRATFOR自体で書かれ，他の処理系に移すのが容易なように工夫されている．

こうしてプレコンパイラを使うことにすると，ユーザの多種多様な要求に応じて数多くの（恐らくは特殊目的の）FORTRAN風言語仕様が必要となり，それらのプレコンパイラもそれぞれ必要となろう．そうなると次にはプレコンパイラを簡単に作成するためのソフトウェアが欲しくなる．これは一種のコンパイラ・コンパイラであり，この場合とくにFORTRANプログラムのジェネレータである．RATFORコンパイラの作成にはコンパイラ・コンパイラYACC (Yet Another Compiler-Compiler) が使われた．今後ともソフトウェア工学の進歩により，実用に耐えるFORTRANジェネレータが生まれることを期待したい．

6. FORTRAN コンパイラの機能

FORTRAN コンパイラはよく使われるだけあって今日では非常によいコンパイラが出現するに至っている．しかし今後ともコンピュータ技術やソフトウェア工学が進歩するとすれば，今日のコンパイラを将来もそのまま使えばよいということにはならないであろう．もちろんコンパイラの機能を増やせば，コンパイラが当然それだけ複雑になることは避けられないが，それは技術の進歩でカバーしなければならない．次にFORTRAN コンパイラがもってもよいと考えられている機能をあげてみる．

(1) エラー・チェックの強化

エラー・チェックを比較的よく行うコンパイラとしては，WATFOR や WATFIV などの教育用コンパイラ[12]がよく知られているが，最近の OS 7-FORTRAN のような大型コンパイラでもエラー・チェックはかなりよく行われるようになっている．しかし今後はさらに徹底したチェッチ機能，さらにはエラー訂正機能も望まれる．

FORTRAN のエラーで厄介なのは，添字の範囲オーバーに伴うもので，これにはハードウェアによるチェック機構が欲しいところである．しかし，やはりチェックしないオプションもないと，たとえば

```
DIMENSION A(123), B(456)
CALL PARAM (A, B)
.....
SUBROUTINE PARAM (A, B)
DIMENSION A(2), B(2)
A(123)=0.0
```

のような（正しく動く）プログラムを書く人は困るかもしれない．しかしこの例は，本来は引数を A, M, B, N の四つにして，サブルーチン PARAM の中で DIMENSION A(M), B(N) と宣言すべきもので，それでないといけないということにしてもよいであろう．

また引数の型の相違によるエラーや FORMAT 文のエラーなども，現在のコンパイラでは実行時にしか分らない(しかも原因がつかみにくい)ことが多いが，これらももっとコンパイル時にチェックすべきであろう．さらには，リンケージ・エディタもとくにエラー検出に関してもっとインテリジェントになってくれないと困る．

(2) オブジェクト・コードの最適化

これは今後のコンパイラでも最重点項目のひとつとして当然行われるであろう．この点については，すでにのべたように，仮想記憶系およびバッファー・メモリをもつコンピュータが増え，アレイ・プロセッサ導入の動きが高まっていることからして，今後はメモリ・ハイアラーキーやアレイ・プロセッサを考慮した最適化が重要になると思われる．

(3) ソース・プログラムの最適化

これの例としては P. B. Schneck らの FORTRAN-to-FORTRAN 最適化コンパイラや FINESSE[15] がある．これでは，最適な形に書き直された FORTRAN プログラムが結果として出力される．したがってコンパイラ（トランスレータ）がうまく作ってあれば，「実行効率を上げるにはプログラムはこう書きなさい」ということをユーザに教える効果も出る．ただそのために，たとえば T00028 といった名前自体に全く意味のない中間変数が導入されたり，プログラムの構造がかえって分りにくくなったりすることもあり，構造化プログラミング的な立場からは，ソース・レベルの最適化には問題も多い．

この種の最適化をさらに徹底させる試みとしては，Guy de Balbine の structuring engine[16]がある．これは構造化されていない FORTRAN プログラムを，do while, do until, do case, do forever などの構文を使って，より明確な構造をもつ拡大 FORTRAN 版に再構成し，それをトランスレータにかけて，よい構造をもつ純 FORTRAN に書き直すシステムである．これは試みとしては面白いが，このエンジン自体が PL/I で実に 30,000 行というぼう大なプログラムになっている点が問題である．

(4) FORTRAN ソース・エディタと文書化用エディタ

先の美化プログラムの機能をさらに徹底させようとすると，結局 FORTRAN ソース・プログラム専用のエディタが欲しくなる．これは独立のエディタとしてもよいが，テキストが FORTRAN のソース・プログラムであるということを意識するという意味で（とくに構文チェックも含める場合）コンパイラとも共通する面はある．FORTRAN エディタではある文字列をすべて修正するときに，それをあるルーチンの中だけに限るとか，コメント行の中は除くとかという指定ができるべきであろう．なお，エディタにはドキュメーテーション（文書化）用のエディタも考えられる．英国国立研究所で ALGOL 用に開発された SOAP (Simplify Obscure ALGOL Programs) では，変数はもとより定数も含めた相互参照リストを作るほか，

POLISH (Paginate Optimally a LISting with Headings) の機能ももっている[17].

7. FORTRAN プログラムの動作解析

FORTRAN プログラムの動作解析に関するものとしては，FORTRAN 文の出現頻度カウント，実行ステップ数解析，エラー統計作成，ライブラリ・プログラム利用頻度カウントなどがあり，その使用目的も，プログラマ個人用のほか，グループ管理者(教師など)用あるいはコンパイラ設計者(またはセンター管理者)用などさまざま考えられる．

(1) FORTRAN 文の出現頻度カウンタ

これは FORTRAN 文の静的解析に相当する．D. Knuth[18] などが作って以来，内外の各所で行われている．われわれのセンターで作成中の改良版では，単なる文の出現頻度のみならず，DIMENSION の次元数，型別，大きさなどの分布も含まれる．とれるデータは主にコンパイラ設計者の設計データとして使われるが，GOTO 文の頻度などは各プログラマの評価を行うのに使えないこともないであろう．

(2) FORTRAN プログラムの動的解析プログラム

これも profiler や FORDAP といった名のプレコンパイラなどの形で内外で試みられている[19)~21)]．普通は各文（各行）の総実行回数を測定してヒストグラムの形で表示する．この目的はもちろんプログラムのどの部分がもっとも時間を食うかをみて，そこを改良するのに使うことにある．とくに繰り返して使うプログラムはぜひとも動的解析をして改良をはかることが望ましい．とれる情報は FORTRAN 処理系の設計・改良にも非常に役に立つ．

(3) エラー統計作成

これはたとえば東大教育用計算機センターの MELCOM 7700 (Sigma 7) の FLAG コンパイラでは，アカウント・データの分析という型で行われている．要するに学生個人ごとおよびクラスごとにコンパイル時にどのエラー・メッセージが何回出たかを出すものであるが，これは教師にとっては自分の教育の反省材料となり，FORTRAN の欠点を知るのにも大いに役立つ．東大の大型センターでは，出力ファイルを調べることにより，エラー・メッセージの統計をとることを時々行っている．しかしこうしたやり方は余りにも間接的であり，将来の FORTRAN 系では，少なくともシステム全体として，どんなエラー・メッセージを何回出したかのカウント位はとれるようにすべきであろう．

(4) FORTRAN ライブラリの使用頻度カウンタ

これは名大大型計算機センターでも行われたことがあるが，東大大型計算機センターでは常時行っている．方法としては，ライブラリ・プログラムを使用するためのマクロ・コマンドの中で，呼ばれたルーチンの名前を主記憶のある領域に積み重ね，一定時間ごとにその領域の内容をファイルに移し，ルーチン名ごとの出現頻度を求めるやり方をとっている．マクロ・コマンドとしては，ルーチンを使うとき，ルーチンをコピーするとき，ルーチンをプリントする（ソース・リストをとる）ときなどのためにそれぞれ別のものが用意してあるので，呼び出された回数，コピー回数，プリント回数などが各ルーチンごとに分るようになっている[22]．ライブラリ・ルーチンはほうっておけばどんどんたまる一方である．そこでこうしたカウンタは，使用頻度の高いものほどよくサポートし，使われないものは切り捨てるといったサポート・レベルをきめるのに決定的に重要である．

8. FORTRAN サブルーチンとプログラム・パッケージ

従来の FORTRAN 処理系はバッチ処理系が多かったため，サブルーチンやパッケージはバッチ処理用に書かれてきた．しかし近年 TSS の利用が高まるにつれ，対話型のルーチン，すなわち実行時に端末からパラメータの値が投入できるようなプログラムの重要性が増している．こうしたプログラムには同時に多数のユーザ間で共用されるものも多い．こうしたことから，システム・プログラミング用サブルーチンとして，たとえば次のようなものも望ましくなってくる．

(1) キーボードからの割込みがあったら制御をある文番号の文に移す．

(2) 引数としてファイル名が与えられたとき，その名のファイルがすでに存在すればそれを割当て，なければその名のファイルをその場で作って割当てる．

(3) プログラムの中で各種コマンドを実行させる．

(4) ユーザ識別名や端末識別名を読みとる．

こうした FORTRAN サブルーチンは高級ミニコンでは比較的よく実現されているが，大型機（とくに国産機）では余りないようである．とくに上記(3)はシステムによっては難しそうであるが，将来のシステムではこうしたルーチンは備わっているべきであろう．(4)は同じプログラムを多数で同時に使うときのアク

セス制御のために必要となる．

FORTRAN の大型パッケージについて，アメリカでみられる傾向は，数学ルーチンや統一グラフィック・パッケージを IBM などのメーカ製のものに頼らず，第三者の専門ソフトウェア会社のパッケージを使う行き方である．たとえば数学ルーチンの IMSL (International Mathematical & Statistical Library)，統一グラフィックス（ディスプレイ，ストレージ・スコープ，プロッタ，マトリックス・プリンタなどを統一）の DISSPLA (Integrated Software Systems 社)，SPSS (Statistical Package for the Social Sciences, SPSS 社)，SIMSCRIPT (Consolidated Analysis Center 社)，DYNAMO (MIT 関連会社) などはその例である．とくに IMSL は数値解析の専門家を集めて基本ルーチンのアルゴリズムの改良を絶えず行っているといわれ，評価が高い．

わが国では基本ルーチンの改良では名大二宮市三教授の労作がある．イギリスでも Harwell Subroutine Library (英原子力委) や NAG (Nottingham Algorithm Group)ライブラリがやはり数値解析専門家の手で整備されている．こうした動きをみると，数学ルーチンの類はメーカの手で個々に一度作られてそれで終りというのでは不十分な時代がきたというべきである．

9. ポータビリティとネットワーク

現在コンピュータの世界では，構内（インハウス）ネットワークやリモート・バッチ網や広域ネットワークというような形で，コンピュータ同志の接続がだんだん盛んになりつつある．こうなってくると，異なるコンピュータ間で最も共通な言語は FORTRAN となり，しかもネットワーク内の単なるファイル転送で行えることでプログラムの交換がやりやすくなる．そこで重要になるのが FORTRAN プログラムのポータビリティである．

ポータビリティをチェックするための手段としては，たとえば基本 FORTRAN あるいは JIS 7000 規格に実際上広く行われている拡張機能 (ENCODE/DECODE など) をつけた程度のポータブル FORTRAN の仕様を定義して，ある FORTRAN プログラムがその仕様に合っているかどうかをテストする検証プログラム (verifier)[23] やプログラムをポータブル FORTRAN に変換する FORTRAN コンバータを作る試みがある．またそうしたポータブル FORTRAN でポータブル FORTRAN ライブラリを作ることも行われる．そうしたライブラリの中には，メモリ使用効率を上げるため，ユーザには見えない形で内部にスタック機構を設け，ダイナミックにメモリ割当てをしている系もある．特殊なサブルーチンとしては，入出力機番の割付をしたり，機械特有の定数（最大整数，最大最小実数，単精度や倍精度の範囲）を呼び出すサブルーチンなどが用意されるのが常である．一方，FORTRAN の方言としてポータビリティを高めるための新機能（仮想配列など）を定義し，それを含めた拡張 FORTRAN を標準 FORTRAN に変換するトランスレータも開発されている[24]．このトランスレータ自体はもちろん標準 FORTRAN で書かれている．

10. FORTRAN プログラムの評価と検証

FORTRAN 関係のツールとしては，以上あげたものの他に，完成したプログラムの質のよさを判定するプログラムや，完成途中のプログラムの論理的な正当性の検証をしてくれるプログラムがあるとよい．こうしたプログラムは FORTRAN についてもすでに試みられているが，その本格的な出現にはコンピュータ科学やソフトウェア工学の一層の進歩を待たなければならないであろう．

一方消極的な検証手段としては FORTRAN デバッガがある．これは TSS で使うもので，FORTRAN のオブジェクト・プログラムの要所要所に文番号で指定してブレーク・ポイントを入れた上で走らせ，そこで引っかかって制御がキーボードに移ったところで，いろいろな変数の値を調べたり，修正したり，あるいはルーチン名などのトレースをとったりするためのプログラムである．オブジェクトに対して，文番号や変数名やルーチン名といったソース・レベルでの記号が使えることで便利になる．

11. 今後の FORTRAN 処理系

以上概観した通り，ソフトウェア技術の進歩により，FORTRAN だけに限っても，ソフトウェア工学的なツールを作る試みはすでに行われている．次の世代の FORTRAN 処理系としては，もはやコンパイラとライブラリ・ルーチン群があるというだけでは不十分である．とくに今後人件費の上昇とハードウェアのコスト・ダウン，さらにはプログラム自体の複雑化の傾向からソフトウェア開発費の比重が増すことを考えると，FORTRAN プログラミングでもその生産性を高めることは非常に重要である．このためには，FORT-

表-2 FORTRAN 関係ツール

FORTRAN ソース・エディタ (I→S, S→S)
FORTRAN 美化プログラム (S→S)
FORTRAN 動的・静的解析プログラム (S→S, S→M)
FORTRAN プレコンパイラ (E→S)
プレコンパイラ・ジェネレータ (E→S)
FORTRAN コンバータ (E→S, S→S)
FORTRAN ソース最適化プログラム (S→S)
FORTRAN 正当性チェック・プログラム (S→M)
FORTRAN 規格チェック・プログラム (S→M)
FORTRAM コンパイラ (S→O)
インテリジェント・リンカ (O→O)
FORTRAN オブジェクト・エディタ (O→O)
FORTRAN デバッガ (O→S)
各種パッケージとサブルーチン
使用頻度カウント・プログラム

I: 入力, S: ソース, E: 拡張言語, M: メッセージ, O: オブジェクト

RAN 処理系の中に表-2に示したようなツールをできるだけ整備することが望ましいであろう．コンピュータ・メーカにとっては，表-2のようなツールのうちどこまでを標準的にサポートするかについて，今後は重大な選択を迫られることになろう．もちろんこうしたものはすべてメーカがサポートする必要はなく，第三者の専門ソフトウェア・ハウスが非常によいものを開発して，ユーザにリースするのでもよい．

最後に本稿をまとめるにあたり，筆者が1975年5月から1976年1月にかけ滞在したベル研究所での見聞から数多くの示唆をえたことを付記する．また東大の和田英一氏や東工大の木村泉氏にご教示頂いたことに対し謝意を表する．

参考文献

1) ソフトウェア・エンジニアリング特集号，情報処理学会誌，Vol. 16, No. 10 (1975).
2) 藤野・柴合：ソフトウェア開発と構造的プログラミング，電子通信学会誌，Vol. 59, No. 1, pp. 32～44 (1976).
3) O. J. Dahl 編（野下他訳）：構造化プログラミング，サイエンス社 (1975, 原著は 1972).
4) C. Bron & W. de Vries: A PASCAL compiler for PDP 11 minicomputers, Software-Practice & Experience, Vol. 6, No. 1, pp. 109～116 (1976).
5) B. W. Kernighan and P. J. Plauger: Software Tools, Addison-Wesley〔338 ページ〕(1976).
6) B. W. Kerninghan and P. L. Plauger（木村泉訳）：プログラミング書法，共立出版 (1976).
7) 石田・村田：超大型コンピュータ・システム，産業図書 (1975),
8) J. G. Rogers: Structured programming for virtual storage systems, IBM Sys. J., Vol. 14, No. 4, pp. 385～406 (1975).
9) L. P. Deutch and B. W. Lampson: An on-line text editor, CACM, Vol. 12, No. 10 (December 1967).
10) 鶴田，武市，和田：Syntax-directed pretty-printer, 第17回プログラミング・シンポジウム報告集，pp. 155～166 (1976).
11) T. Watanabe and F. Yamamoto: Extran-a top-down programming system, IFIP 74, Vol. 2, pp. 213～217 (1974).
12) B. W. Kernighan: RATFOR-a preprocessor for a rational FORTRAN, Software-Practice & Experience, Vol. 5, No. 4, pp. 395～406 (October/December 1975).
13) 土居他：実習用 FORTRAN コンパイラ CAFT 作成上の問題点と効率，pp. 111～122 (1976).
14) P. B. Schnech and E. Angel: A FORTRAN to FORTRAN optimizing compiler, Comp. J., Vol. 16, No. 4, pp. 322～330 (1973).
15) 山田ほか：広域プログラム最適化理論とその周辺〔5-1〕, bit, Vol. 8, No. 1, pp. 71～77 (1976).
16) Guy de Balbine: Better man-power utilization using automatic restructuring, Proc. of NCC, Vol. 44, pp. 319～327 (1975).
17) R. S. Scowen: Babel and SOAP-Applications of extensible compilers, Software-Practice and Experience, Vol. 3, No. 1, pp. 15～27 (1973).
18) D. E. Knuth: An empirical study of FORTRAN programs, Software-Practice and Experience, Vol. 1, pp. 105～135 (1971).
19) 藤村，牛島：FORTRAN プログラムの動的解析システムの移し換えについて，第17回プログラミング・シンポジウム報告集，pp. 99～110 (1976).
20) D. Ingalls: The execution time profile as a programming tool, in R. Rustin ed., Design and optimization of compilers, pp. 107～128, Prentice Hall (1972).
21) E. C. Russel and G. Estrin: Measurement based automatic analysis of FORTRAN programs, Proc. of SJCC, Vol. 34 (1969).
22) 栃木：プログラム・ライブラリの使用頻度カウンタ，東大大型計算機センター年報，5号，pp. 63～70 (1975).
23) B. G. Ryder: The PFORT verifier, Software-Practice and Experience, Vol. 4, No. 14, pp. 359～377 (1974).
24) D. E. Whitten and P. A. D. de Maine: A machine and configuration independent FORTRAN-Portable FORTRAN (PFortran), IEEE Trans. on Software Engr., Vol. SE-1, No. 1, pp. 111～124 (1976).

（昭和51年5月18日受付）
（昭和51年5月29日再受付）