解　説

プログラムの移植について†

藤田晨二†† 葛山善基††† 川原洋人†††

1. まえがき

ある機種上で開発されたソフトウェアをほかの機種上でも走行させるというソフトウェア移植の必要性は，計算機が出現した当初から叫ばれていた．そして，ソフトウェア移植の容易化は，高級言語の標準化という形で具現されてきている．しかしながら，標準化された高級言語で記述されたプログラムといえども，数%のオーダで互換性のない部分を含み，それらが移植の実作業レベルで障害となることが多い．(筆者らの経験した移植対象プログラムの中には，内部表現形式の差違が原因で移植後のプログラムがうまく動作せず　結局それを正しく動作させるために2人月以上の工数をかけた例がある．これにより修正したステップ数はたった数ステップである．)．

具体的なシステム間におけるソフトウェア移植の事例については多く発表されており[1]，また，ソフトウェアに移植性をもたすことについては，たびたび論じられその方策についても紹介されている[2)~4)]．

本稿では，移植の障害となる機種間の相違点を主たる起因元によって分類することを出発点として，各々が，移植作業にどのように影響するか，また，ソフトウェアの移植性を高めるさまざまな方策によってどのように解決されるかについて整理することを試みた．

ソフトウェアの移植には，プログラムの移植とデータベースのような大量のデータファイルの移植があるが，本稿においてはプログラムの移植に焦点を絞ることとした．

2. 移植の必要性

2.1 既存プログラムの移植の必要性

表-1　プログラムの新規開発工数と移植工数との比較[1)]

(a) プログラムの新規開発時の工数

新規開発	200～400 ステップ/人月

(b) 移植工数

移植プログラムの記述言語	移植工数
高級言語	6K～10K ステップ/人月
アセンブラ (ただし，移植先は高級言語とする)	400～2K ステップ/人月

注) 本表の移植工数は，移植条件の整備されていることを前提としている．

近年，ハードウェア技術の進歩により，コスト性能比のよい計算機が数多く開発されている．そのような状況のもとで，(1)処理能力の向上[5)]，(2)使用計算機の統合による設備費，保守費の削減[6)]，(3)コンピュータの利用形態の高度化[7)]，等を目的としたコンピュータのシステム移行が行われてきている．

システム移行を行う場合，ハードウェア，オペレーティングシステム，言語プロセッサ，サブルーチンライブラリ等は，メーカが提供するものを利用できるが，ユーザ固有のプログラムについては，新規に開発するか，従来使用しているものを流用するかの選択に迫られる．多くの場合は，従来のプログラム機能で十分であることと，新たに作成することは作成期間が長くなり，コスト高になること(表-1参照)から，プログラムの移植による流用が選択されることが多い．

また，システムの更改に限らず，ある機種上で作成されたプログラムを有用性が高いがゆえにほかの機種へ移植するという例も多い．

2.2 移植を考慮したプログラム開発の必要性

たとえば，構造解析，回路解析などの高度な専門技術を必要とするソフトウェアに対する要求が増大しつつあるが，それは，機種とは独立な高級言語の標準化の動向と相ま　て，計算機メーカとは独立なソフトウェアハウス，研究機関などによるプログラム作りを促進している．これらの組織においては，所要の投資効

† On Program Portabilities by Shinji FUJITA (Cross-Industry Application Programs Department, EDP Systems Support Division, Nippon Electric Co., Ltd.) Yoshiki KATSUYAMA and Hiroto KAWAHARA (Yokosuka Electrical Communication Laboratory, N. T. T.).

†† 日本電気(株)
††† 日本電信電話公社横須賀電気通信研究所

表-2 高級言語記述プログラム移植時に問題となる機種間の相違

タイプ	大項目	小分類	小項目	説明および例
I	言語	(a)	構文の相違	同一機能を表現するシンタクスの相違
		(b)	機能の相違	サポートしている文の種類，および組込関数，基本外部関数の機能的な相違 etc.
II	CPU	(a)	語長(表現範囲)	整数，浮動小数点数の表現可能範囲の相違*
		(b)	語長(精度)	浮動小数点数の表現精度，演算精度の相違*
		(c)	語長(文字数)	1語に詰められる文字数の相違**
		(d)	内部表現	文字，整数，浮動小数点数の内部表現の相違
		(e)	その他	境界調整用のダミー域の長さと有無，割り込み条件の相違 etc.
III	ファイル管理	(a)	形式	扱い可能なファイル形式***，レコード形式***，レコード長制限値の相違 etc.
		(b)	アクセス	装置番号の指定可能範囲，FORTRAN 順序番号の有無，標準入出力装置機番の相違 etc.
IV	メモリ管理	—	—	アドレス空間の実メモリへのマッピング方式の相違(仮想記憶方式の相違など)，オーバレイ制御方式の相違 etc.
V	操作管理	—	—	JCL (Job Control Language) および端末コマンド言語などの相違

注) * 精度を桁数で指定できる言語においてはこの問題は緩和される．
** 長さを指定できる文字属性のある言語においてはこの問題は緩和される．
*** FORTRAN プログラムでは直接意識する事はほとんどない．

果を得るために，1つのプログラムを多数のユーザに利用させることが必要となり，複数機種で容易に走行するような，移植性の高いプログラム作りが望ましい．

3. 移植上の問題点

高級言語で記述されたプログラムの移植時に問題となる機種間の相違[9),10)]を主としてFORTRANについて，表-2に分類して示した．大項目は主たる起因元により分類した．表-2の各タイプについて，以下に補足説明しよう．

(1) タイプI—(a) シンタクスの相違

例1に，同一機能の直接アクセス入出力文のシンタクスが機種によって相違する場合を示した．また，文の出現順序規定の相違や，同一機能の組込み関数，基本外部関数の名称の相違もタイプ I—(a) に含めることとする．

例1 タイプ I—(a)

READ $(u, f, \mathrm{REC}=i)\ k$

⇕

READ $(u' i, f)\ k$

(uは装置番号指定子，fは書式指定子，iはレコード番号指定子，kは入力並びである．)

(2) タイプ I—(b) 機能の相違

たとえば，ENCODE, DECODE 文の有無や関数機能の差がある．このタイプの場合は，修正箇所の抽出は，コンパイラのチェック機能等を用いることにより，簡単な場合が多いが，移植先の機種にない機能の代替設計などを検討する必要があり，機械的な修正では対処できない

(3) タイプ II—(a) 語長 (表現範囲)

表現可能範囲の相違は，オーバフロー，アンダフローの問題となって現われる．

(4) タイプ II—(b) 語長 (精度)

語長の長い計算機から語長の短い計算機へプログラムを移植した場合によく精度の悪化が問題になる．中には，くり返しにおいて，全く値が収束しなくなるなど，論理的なパスも変化する場合もある．

(5) タイプ II—(c) 語長 (文字数)

たとえば，単精度実数型の変数に格納できる文字数は，CDC の場合 10 文字 (60 ビット)，IBM の場合 4 文字 (32 ビット) と異なる．

例2 タイプ II—(c)

```
DATA X/10 HABCDEFGHIJ/
```

(Xは単精度実数型変数)

したがって，例2のような DATA 文の結果は，CDC と IBM の FORTRAN で異なる．

(6) タイプ II—(d) 内部表現

内部表現を意識したプログラム記述の例として，ソーティング時によく用いられる，文字列を格納した整変数の間での大小比較がある．この場合，内部表現の相違により結果の異なることがある．たとえば，JIS コードと EBCDIC コードでは，数字と英字の大小関係が逆であり，下記の FORTRAN プログラムのステートメントの実行結果は，異なる．

例3 タイプ II—(d)

```
      LOGICAL C
      DATA A, B/4HABCD, 4H1234/
            ⋮
      C= A.GT. B
```

(EBCDIC コードを用いているマシン上ではCの値は，.FALSE. であり，JIS コードを用いているマシン上ではCの値は，.TRUE. となる.)

(7) タイプ Ⅱ—(e)

境界調整用のダミー域が影響する例として，例4にCOMMON による変数結合例を示す.

例4 タイプ Ⅱ—(e)

```
SUBROUTINE SUB 1
REAL A
DOUBLE PRECISION X
COMMON /Z/A, X
      ⋮
SUBROUTINE SUB 2
REAL A, B, C, D
COMMON /Z/A, B, C, D
```

機種A 機種Bは，共にバイトマシンで単精度変数が4バイト，倍精度変数が8バイトとする.

機種Aでは，倍精度変数は8バイト境界におくものとし，機種Bでは，境界調整を行わないとする. このとき，機種Aでは，XとC，Dが同一領域であり，機種Bでは，XとB, Cが同一領域となるので，機種A, B で参照関係が異なる.

(6), (7) のような問題は，発生しても，原因がそれと容易に分析できず，発生箇所の抽出も難しいので，移植上の厄介な問題である.

(8) タイプ Ⅲ

このタイプに属する相違点のうち，変換時の修正が難しい例を例5に示す.

例5 タイプ Ⅲ

```
WRITE (J) (X(I), I=K, KK)
```

上記のようなステートメントで出力されるレコード長が移植先の機種のレコード長制限を越える場合には，修正を行う必要がある. ここで，出力されるレコード長は，実行時に決まるKとKKの値によって，変化すること，レコードを単純に制限内に入るように複数に分割して入出力を行うようにプログラムを変更する方法をとると，BACKSPACE 文を実行したときに，レコードポインタが分割された単位分しか戻らず，移植元の機種での動作と異なってくることなどを考えると修正が難しい.

(9) タイプ IV, V

タイプⅣの問題は，実行できるプログラムサイズの相違，オーバレイされるセグメントのアドレス空間上の位置を意識したプログラム（たとえば，オープンコア[11]と呼ばれる手法を用いたプログラム）を変換するときの問題などとして現われる. この場合には，プログラム構造および制御方式の検討を行わなくてはならない. タイプVについては，JCL コンバータなどによって解決が図られる例もある. タイプIV, V についてはプログラム記述上の修正では対処が難しい.

以上述べてきた(1)〜(9)の相違点を緩和し，プログラムの移植を容易にする方策がいろいろ考えられている. それらは，大きくわけて，移植を考慮しないですでに作成されたプログラムに対する方策と，最初から多機種で走行させることを前提に，移植しやすいプログラム作りを行うための方策とがある. 次章以下で，それらのうち言語レベルで行われる方策について見てゆくこととしよう. (したがって，タイプ IV, V に関する相違については以下の稿で言及しない.)

4. 既存プログラムの移植方法

4.1 移 植 作 業

ここでは，既存プログラムを移植するときの作業[4),8)]について，(1)〜(6)の項目に分けて述べる. このうち，(2)〜(6)は，移植するプログラムごとに必要な作業である.

(1) システム環境の相違の調査

前章に述べた移植上問題となるシステム間の相違点をあらかじめ調査する. コンバージョンを行う前に機種間の相違をすべて細くチェックアウトすることは困難であるので，移植作業を進める中でその知識を深めてゆくのが一般的である.

(2) 移植対象プログラムの準備

移植元の機種で，作成されたプログラム（ソース）ファイル，テストデータファイルを移植先の機種のファイルへ変換する. その際必要に応じて，コード変換，ファイルの形式の変換を行う. また，移植後のプログラム動作確認のために移植元の機種でのテストデータの走行結果も入手する必要があり，さらにデバッグに役立つプログラムドキュメントの入手が望ましい.

(3) 移植対象プログラムの分析

移植対象プログラムの分析には種々のレベルがある. それらのレベルには，

(i) プログラム論理を，すべて解読し変換上必要

な項目を洗い出すという一番詳細なレベル.

(ii) 変換上問題となりそうな項目，たとえば，入出力文の使用状況，システム関数，システムサブルーチンの使用状況，語長の相違や，演算精度の相違の影響度など今後の作業に重要な影響を与えそうな点について解析するレベル.

(iii) プログラム規模，記述言語，およびプログラムの動作環境（入力，出力，必要メモリ量，必要ファイル量）など，すぐに調査可能な範囲にとどめ，それ以外の問題については，以後の変換作業の中で発生ごとに解決していくことを考えるレベル.

などがある.(i)のレベルは，移植作業にプログラム作成者が加わる場合には自然に達成されるがそうでない場合は，作業量の増大を招くため，一般に採用されない.(ii)または(iii)のレベルが一般的である.

(4) スケジューリング

プログラムの分析結果にもとづいて作業計画を立案する必要がある．プログラム規模と記述言語からプログラムの修正のための工数が予測されるだろうし，プログラムの試験のための工数は，テストデータの本数（これは，変換結果の品質を左右する.）出力結果の量（形態）から，予測されるだろう．また，プログラムの分析の結果，新たに必要なツールがあれば，その作成工数も考慮する必要がある．これらから，作業期間，必要マシン時間などが予測される．これらの予測は，今までの経験の平均値として割り出されるがプログラムによって問題の発生のバラツキが大きいために，個々のプログラムについてみれば移植作業が予定通り行われるということは，あまり期待できない.

(5) 変換作業

変換作業としては，コンパイラのチェック機能を利用した言語間シンタクスの相違の吸収をはじめとし，システム間の相違を吸収するための，プログラムの変更作業を行う．さらに，コンパイラ，リンケージエディタなどを利用した実行形式プログラムの作成を行いテストデータを用いてそれを走行させその走行結果を移植元の機種での走行結果と比較し，変換結果の確認を行う．その結果が不良ならば，デバッグ作業を行う必要がある．コンパイラは，変更の必要なステートメントをすべてチェックアウトするわけではないので，チェックアウトもれとなったものは，このデバッグ作業の中で吸収する必要がある．走行結果で精度が問題となったときは，プログラムの倍精度化の作業を行う必要も出てくる．倍精度化は一般には，宣言文の変更，追加による変数の倍精度化，関数や定数の倍精度化などによって行われるが，このような修正を機械的に行うと倍精度化された変数と倍精度化されない変数（整数型変数など）が結合されている場合，文字が格納されている場合などには誤動作をおこしやすい．また，修正ステートメントが大量にある場合には，修正ミスによる作業の遅延も無視できない.

(6) プログラム評価と保守体制の構築

(5)で行った変換結果についてプログラムの走行結果の評価(処理時間，演算精度などの走行結果の評価)をまとめ，今後，新環境上で使用する場合の使用手引書，保守手引書などのドキュメントを整備する．また，今回の移植作業中に発生した問題点などを整理し，次回の移植に反映させることなども有益である.

4.2 コンバータの効果

移植作業を容易化するためのツールにはいろいろのものが考えられるが，ここでは言語上の対策という観点からコンバータについて述べる．コンバータは，ある機種の言語仕様で書かれたプログラムのステートメントを，移植先の機種で，正常にコンパイルされ，動作するように自動的に修正するツールである．コンバータの出力は，移行先の機種上のコンパイラで処理される．コンバータの処理は，ソースプログラムの構文解析にもとづく変換が主である．つまり，タイプI—(a)に対して有効なツールであり，また，タイプI—(a)以外に対しては，シンタクスの解析で抽出できるもの（たとえばタイプI—(b)の大部分）について，警告メッセージの形でユーザに通知することになる.

タイプII—(b)の精度に関しては，精度の劣化を防ぐために機械的に宣言文，関数，数値などの倍精度化を行うコンバータもあるが[14]，4.1(5)に述べたと同様の問題はある.

タイプIIIのうち論理解析を必要とする問題（例5参照），およびタイプII—(a), (c), (d), (e)については，一般にコンバータでは扱えない.

コンバータの例として，電電公社の DEMOS-E でサービスされている FSCONV を紹介[12]する．FSCONV は，IBM の FORTRAN H 拡張，または，HITAC の VOS 最適化 FORTRAN から DEMOS-E FORTRAN に変換するものである．これらの機種間では，語長が等しいので，タイプII—(a), (b), (c)の問題が生じない.

FSCONV では，入力プログラムをシンタクス定義にもとづいて構文解析木の形に構文解析し，変換は変

換定義にもとづいて構文解析木の各々の節に対するオペレーションとして行われる[13]．シンタクス定義および変換定義は，差し換え可能なテーブルで記述されているので将来的には，その差し換えにより対象機種の拡張，シンタクスに着目したプログラム解析ツールや，種々の変換ツール（たとえば倍精度化ツール）への改造が可能な形になっている．同じような処理形態のコンバータとして TAMPR[14] があり，このような発展性のある構成方式をとるのが今後の移植作業を展望したときには，望ましいと考えられる．

筆者らの経験によると，コンバータの使用効果は，手作業で修正していたステートメントを，大部分自動化すること，(副産物として手作業による修正時のミスが削減されること）および，コンパイラで，チェックできない修正の必要なステートメントを前もってチェックアウトまたは自動修正することによるデバッグ作業の短縮である　すなわち，コンパイラでチェックアウトできないステートメントの修正は，コンバータがなければ，実行時のデバッグに委ねられるわけで，そのようなステートメントに遭遇するたびに，コンパイル・リンク・ラン，および結果の確認作業の繰り返しになり，作業量の増大を招くわけである　それらの中には，たとえば，基本外部関数の名称の相違などのようなタイプ Ⅰ—(a) のものも含まれる．

5. 移植を前提とするプログラムの作成法

初めから，複数機種上で，走行可能な移植性を持つことを目標にプログラムを作成する時に，考えられる方策について以下に述べる[4),15)]．

(1)　言語仕様の共通部分でのプログラムの記述

この方法は，あらかじめ，動作する機種を，複数設定し，それらの機種の言語仕様の共通な部分でプログラムを記述するものである．

シンタクスの共通な部分，機能の共通な部分のみを用いてプログラムを記述することにより，タイプⅠの問題の発生が避けられる．また，タイプ Ⅱ—(d), (e), タイプⅢについても，その相違による問題が発生しないようにプログラム記述制限を設ける．

タイプ Ⅱ—(b) の問題については，対象とする機種の中で語長の一番短く，一語長での精度が最も低い機種で，十分な精度を保証するように，関数，変数，数値の精度を上げて，プログラムを記述し，精度の高い機種でもそのまま使用するという共通化が一応考えられる．

その場合語長が長く精度の高い計算機において，メモリ使用効率，実行速度が悪化する．たとえば，語長の長い計算機で，倍精度演算をハードウェア命令でなくソフトウェアシミュレーションで行っている例がある．したがってこのような精度の問題に関する共通化は，適当ではなく，解決策とはならない．またタイプ Ⅱ—(c) については，一語に詰める文字数を一番少ないものへ共通化することが考えられる．この場合語長の長い機種において，メモリ効率が悪くなるが，文字列に対するオペレーションを頻繁に行わないプログラム（たとえば，科学技術計算プログラム*）においては，全体としての効率は落とさないので，良く採用される．

タイプ Ⅱ—(a) については，プログラム内の値を共通範囲内に制限することも考えられるが，次に紹介する PFORT のような扱いもある[16)]．

この方法を採用するための条件として，(1)共通言語仕様が十分な広さ，すなわち機能を備えていること，(2)共通言語仕様がわかりやすいものであること，などがあり，(2)については，チェッカなどがあることが望ましい．

共通言語仕様の適用例として，ベル研究所の共通言語仕様 PFORT (Portable FORTRAN) がある[16)]．PFORT は，IBM 360/370, UNIVAC 1100 シリーズ，Honeywell 6000 シリーズ，PDP 11 などの FORTRAN に対し移植性のあるプログラム作りをサポートしている．PFORT には，PFORT Verifier と呼ばれるチェッカがあり[17)]，共通言語仕様内で記述されているかどうかのチェックが可能である．また，機種固有の定数（たとえば，標準入出力機番，単精度実数，整数などの最大値，最小値）を，関数として参照できる機能も用意されている．たとえば，I1MACH という整数型の関数が用意されており，I1MACH (9) は対象とする機種の正の整数の最大値を返す．(すなわち，I1MACH という関数は対象機種ごとに，別々に記述され，各機種固有の値を返す．)この機能を用いることにより，タイプ Ⅱ—(a) の問題に関し，下記の例に示すようにプログラムの共通な記述が行える．

例　オーバフローチェック

I1 と I2 を正整数とし，I1 と I2 の積を求める前にその値がオーバフローするかどうかチェックする．

* すでに述べたように一語の文字数が問題となるのは，主として，FORTRAN であり，FORTRAN で書かれたプログラムは，科学技術計算が多い．

```
      IF (I1 .GT. I1MACH(9)/I2)
     * GO TO 〔オーバフロー時の処理〕
      I3=I2*I1
```

なお，PFORT では，精度の問題をサポートしていないので，精度の問題が生じるプログラムは，単精度プログラム(語長の長い計算機向き)，倍精度プログラム(語長の短い計算機向き)の，両方を作成する必要がある．また，タイプII—(c)の問題については，DATA 文において，変数に一文字しかセットしないように定めている[17]．

(2) コンバータを考慮した共通言語仕様

前節(1)で述べた移植性を持たせる機種間に，コンバータが存在する場合には，プログラマの負担を，大分，軽減することができる．

この方法は，機種Aから機種Bへのコンバータがある場合，機種A上で作成するプログラムを記述する際に機種Aと機種Bに共通な言語仕様に加えてコンバータで機種Bへ自動的に変換できる部分（タイプ I—(a)）を，使用しても良いとするものである．すなわち，その分だけ“共通言語仕様”の範囲が広くなるわけである．また，タイプI—(b)などのコンバータでチェック可能な相違については，コンバータがチェッカの役割を果たす．したがって，プログラマが，十分気を付けなければならない部分は，タイプII—(a),(c),(d),(e)，およびタイプIIIのうちコンバータでチェック不可能の部分だけとなり，負担が大分軽んぜられる．

タイプII—(b)の精度問題については，もし，使用するコンバータに倍精度化の機能がある場合には，解決可能である．ただし，コンバータの倍精度化は形式的に行われるのでその結果，論理的な誤りを起さぬようにプログラム記述規則を設ける必要がある．

コンバータを考慮した共通言語仕様の例として，前述の FSCONV を利用したものが筆者らの電気通信研究所で行われている．そこでは特に，タイプII—(d),(e)*，タイプIIIの相違点に起因する厄介な問題点が発生しないように記述規則を定めている．その記述規則には，文の順序関係や組込み関数の名称の相違などタイプIのシンタクス上の相違に関するものは含まれておらず，プログラマは注意を払う必要がないので，プログラム記述上の負担は少ない．

(3) 共通言語コンパイラ

(2)で述べたコンバータを考慮した共通言語仕様の拡大の範囲はタイプI—(a)の問題に限られている．さらに，タイプI—(b)の問題をも解決する方式として，共通コンパイラの方式がある．

共通コンパイラの方式とは，同一言語で記述されたプログラムを対象機種ごとのオブジェクト・コードに変換する対象機種用の各コンパイラを共通に開発する方式である．

この方式によれば，言語仕様は同一であるので，タイプI—(a),(b) の問題はなく，さらに，コンバータとコンパイラを2重に走行させる必要はない．しかし，その他の問題については，(2)と同様である．

この一例としては，DEMOS-E の一ライブラリであるマイクロプロセッサ・ソフトウェア開発支援システム (PMP) の一プログラムである PMP-CE 汎用クロスコンパイラ[18]がある．

この汎用クロスコンパイラは PL/I に類似した各機種共通の言語をもち各種のマイクロプロセッサのプログラムのコンパイルが可能である．また，このコンパイラは，中間言語を生成する部分までを共通化されており，コンパイラの機種依存部がジェネレータ方式により生成される構成となっている．現在，PMP-CE は，マイクロプロセッサを対象としたシステムプログラム記述用言語のコンパイラであるが，この考え方は，ミニコンおよび汎用大型計算機の FORTRAN, COBOL 等の高級言語コンパイラにも適用できるものである．

また，PASCAL コンパイラのように走行マシンを特定の計算機に限定せず，コンパイラ自身を移植する共通コンパイラがある．このコンパイラは，中間言語を生成する部分までを共通化するとともに，その OS インタフェースを局所化，仮想化し，複数機種上で走行可能とする方式である．

(4) プリプロセッサによる移植性

本稿で述べるプリプロセッサとは，そのプリプロセッサ入力用に定められた記述規則を持ち，それで記述された1つのプログラムを入力として，機種別にその機種のコンパイラへ入力可能なソースプログラムに変換して出力するものである．すなわち，プリプロセッサの利用者は1つのプログラムを書くことにより，複数の機種別のプログラムをプリプロセッサから得られるわけで，その意味でその複数の機種へ移植性のあるプログラム作りを可能としている．入力プログラムの記述規則，すなわちプリプロセッサへの変換指示の方

* 記述規則の対象とする機種間，すなわち，FSCONV の対象とする機種間では，語長が等しいので，タイプ II-(a),(b),(c) の問題は生じない．

法としてはマクロ記述による方法[19]，機種ごとに異なるステートメントに対し，異なる形をすべて記述し，プリプロセッサがそのうち1つを選択出力できるようにする方法などが考えられる．

プリプロセッサによる方法では，タイプⅠ—(a)の問題を解決し，さらに対象機種の精度，語長に応じて出力を変化させることができるのでタイプⅡ—(b),(c)の問題も解決できることが特長である．プリプロセッサの記述規則の中には，共通言語仕様の所で述べたような記述制限を設けプリプロセッサで処理しない所，しにくい所をカバーするのが一般的であり，その場合には，タイプⅠ，Ⅱ，Ⅲのほかの問題も解決する．

また，プリプロセッサは，プログラマが，修正箇所，修正方法を指定するので，一般にそのようなことが期待できないコンバータに比べずっと構成が容易になる．反面，プログラマにとっては，プリプロセッサの処理とその出力プログラムを意識せねばならず，その点負担が重くなる．

このようなプリプロセッサの例として，アメリカのIMSL社のIMSLプリプロセッサ*がある[20]．

IMSLプリプロセッサにおけるステートメントの変換指定は，プログラム変更箇所を直接コメント文で指定する方法である．

例　入力ソースプログラムの記述例

```
C$ IF (SSIG 10. LT. 10) 1 LINE
      DOUBLE PRECISION SUM
        ⋮
      SUM=0.0
      DO 10 I=1, N
   10 SUM=SUM+X(I)
```

上記の例はC$と記したコメント文により単精度の精度が，10桁以上ならば次のステートメントを，コメント化（無効化）しなさいという指示であり，IMSLプリプロセッサはそれ自身のもつマスタファイルに格納されている対象機種の単精度の精度を調べ，コメント化するかしないか決定する．

そのほかに，IMSLプリプロセッサでは，定数の設定機能，倍精度化機能などがあり，以下にそれらを簡単に紹介する．定数の設定機能を用いれば，タイプⅡ—(a)の問題に関し，PFORTの場合と同様な解決が可能である．

（1）　定数の設定機能……機種固有の定数（たとえば表現可能最大単精度実数値）をマスタファイルから読み出しプログラマの指定箇所にセットする機能である．また，π（円周率）などの定数もマスタファイルから単精度，倍精度などの形で取り出し，ソースプログラム内の指定箇所にセットできる．指定箇所は，コメント文により示される．

（2）　倍精度化の機能……指定により一定の規則で，プログラム全般にわたり変数，配列の倍精度化，SIN, COSなどの基本外部関数，組込み関数の参照の倍精度化を行う．プログラマは，一定の記述規則（たとえば，型宣言文でR␣EALなどとブランクを挿入してはならない．）に従ってプログラミングすることにより本機能を活用できる．

プログラマは，プリプロセッサの入力プログラムを記述することにより，複数の対象機種の必要な精度を持つプログラムが得られる．IMSL社では，IMSLプリプロセッサを用いることにより，IBM, UNIVAC, CDC, Burroughsなどの7社以上の機種に対して移植性をもつプログラム作りに成功している．

6. あとがき

本稿においては，高級言語で記述されたアプリケーションプログラムの移植について述べてきた．これらは実際のシステムにおいて利用されていることであるが，さらに，研究的なレベルにおいては，すでにOSの移植等が試みられている[21]．これらの評価については，今後に期待したい．

最後に，日頃ご指導頂く横須賀電気通信研究所データ通信研究部戸田巌部長ならびに飯村二郎統括役に感謝致します．

参考文献

1) コンピュータ・システムの拡張と変換，ビジネスコミュニケーション，'71, 7-'72, 12.
2) 松下，山崎，丹下：プログラム・トランスファラビリティ，情報処理，Vol. 16, No. 10, pp. 871-879 (1975).
3) 前田：ポータブル・ソフトウェア作り，bit, Vol. 10, No. 12, pp. 4-10 (1978).
4) ソフトウェア産業振興協会：プログラム変換系，ソフトウェア・エンジニアリングに関する調査研究，53-S 030-1 (1978).
5) Woods, J. B.: Converting from the 370 to the 470, DATAMATION, Vol. 23, No. 7 (1977).
6) Contemplating Conversion? An inside story on replacing IBM, DATAMANAGEMENT Vol. 16, No. 4 (1978).

* 文献20)ではIMSLコンバータと記述されているが本稿では，IMSLプリプロセッサと称することにする．

7) Francois, J.: Converting a Maxi Package to a Mini System, DATAMATION, Vol. 22, No. 11 (1976).
8) Oliver, P.: Guidelines to Software Conversion, ACM, SIGCIM Review, Vol. 8, No. 1 (1979).
9) ソフトウェア産業振興協会: FORTRANプログラム移植の手引書, ソフトウェア・エンジニアリングに関する調査研究 53-S 030-2 (1978).
10) Brian, T. Smith: Fortran Poisoning and Antidotes, Portability of Numerical Software, Lecture Notes in Computer Science 57, Springer-Verlag (1976).
11) HITAC 総合構造解析システム ISAS —第1分冊— 機能編 8080-7-028 (1978).
12) 日本電信電話公社: DEMOS-E FSCONV 説明書, DEMOS-E 2491 (1980).
13) 中庭他: メタ言語記述を用いた FORTRAN コンバータ, 昭和54年度信学会総合全国大会.
14) Dritz, K. W.: Multiple Program Realizations Using the TAMPR System, Portability of Numerical Software, Lecture Notes in Computer Science, Springer-Verlag (1976).
15) James M. Boyle: Mathematical Software Transportability Systems-Have the Variations a Theme? Portability of Numerical Software, Lecture Notes in Computer Science 57, Springer-Verlag, pp. 305-360 (1976).
16) Fox, P. A.: PORT-A portable Mathematical Subroutine library, Portability of Numerical Software, Lecture, Notes in Computer Science 57, Springer-Verlag (1976).
17) Ryder, B. G.: The PFORT Verifier, Software-Practice and Experience, Vol. 4, pp. 359-377 (1974).
18) 日本電信電話公社: DEMOS-E PMP 説明書その 10 DEMOS-E 7301 (1980).
19) 鳥居他共訳: マクロプロセッサとソフトウェアの移植性, 近代科学社(1977).
20) Aird, T. J.: The IMSL FORTRAN Converter: An Approach to Solving Portability Problems, Portability of Numerical Software, Lecture Notes in Computer Science, Springer-Verlag (1976).
21) Johnson, S. C. and Ritchie, D. M.: Portability of C Programs and the Unix System, B. S. T. J., UNIX Time-Sharing System, pp. 2021-2048 (1978).

(昭和55年6月3日受付)