解　説

UNIXの日本語化の実現方法†

小　野　芳　彦††

1. はじめに

計算機業界では，OSの標準化によって現在の混沌を救おうという期待が高まっており，その最有力候補としてUNIXの名があがっている．また，日本では，UNIXでアルファベットに加えて日本語も表記・処理できるようにするために，漢字を含んだ新しい文字セットを導入するという拡張が個々に進んできている．UNIXの日本語化の標準については，UNIXの配給元であるAT&Tを初め，通産省後援のΣプロジェクト[1]や日本ソフトバンク後援の32ビット・パソコンUNIX研究会などで審議されてきており，現在も審議や具体化が続けられている．本稿では日本語を取り扱う拡張をUNIXに施すことにともなう技術的問題とその実現方法，および，国際的なUNIXの標準化の一環としての日本語化の問題点について論説する．

UNIXをOSとする計算機の内外のメーカは，日本においてそれぞれ独自に日本語化に取り組んできていた[2]．各国語化という面では対応が遅れていたAT&Tは，単なる日本語化ではなくアジア・ヨーロッパ諸言語のための拡張も含んだUNIXの国際化という方針を打ち出し，その最初の応用としての日本語化に取り組み，1985年に第1版のJAE (Japanese Application Environment) をリリースした．

JAEの仕様は，AT&Tインターナショナル・ジャパン社（東京）から上記国際化の一環としての諮問をうけて日本語UNIX諮問委員会（大学とNTT，富士通，日立，三菱，NEC，沖，東芝，AT&Tユニックス・パシフィックの委員より構成，委員長石田晴久東大教授，以下委員会という）が審議した提案書[3]におおむね基づいている．筆者はその委員会での立案・審議の場に参画した一員である．

本稿は，諮問以降にまとまった筆者の個人的意見（委員会やAT&Tの見解とは異なる面もある）も含めて，上記提案書やJAEに盛り込まれた日本語化における指針となるものを論説したものである．本稿では，委員会の審議の中の大きな争点であった問題点を取り上げる．

2. 日本語を扱う基本方針

UNIXが発展したその背景には「小さく作って大きく使う」というUNIXの設計哲学がある[4],[5]．筆者はこの設計哲学がUNIXの本質であると感じており，UNIXの日本語化といった問題の解を得る上で指針となるものであると考えている．

「小さく作る」ということは，その作られるエレメントが持つ機能を単能に限ることに徹するということである．日本語処理を導入するにおいてもこの方針をできるだけ育てる方向に向かうべきである．

「大きく使う」ということは，単能のエレメントの組合わせを容易にするインタフェースが重要ということであり，このようなインタフェースの単純化に合わせて日本語処理のための拡張がなされなければならない．

さらに，この組合せるという中には，同じような機能を分散させず一カ所にまとめておくという意味合いもある．たとえば，日本語文字を含んだテキストの処理のためのコマンド類はすべてASCII文字のためのものとは別のセットにし，それぞれ独立に利用するといった方法はこの哲学に反するものといえよう．

UNIXらしさを示すもの，UNIXを成功させたものとして次の具体的なものがあげられる．

（1） 均質・柔軟なファイルシステム

（2） パイプ，フィルタ，リダイレクションなどの単能部品とその組合わせ方法

（3） Cによるプログラミング

（1）と（2）を勘案すると，たとえば日本語のための文字セットについて次のようなことがいえる．

たとえば，ファイルが日本語文字を含むものであるか否かの区別やどのような文字コードであるかをファイルの種別としてファイルのディレクトリ（あるいは

† An Implementation Methodology of Japanese Facilities Provided for the UNIX System by Yoshihiko ONO (University of Tokyo).

†† 東京大学理学部情報科学科

iノード）に持つようにし，それをプログラムが認識して対処するという方法が考えられる．しかし，パイプラインにはディレクトリやiノードがないから，そのような方法は(2)の UNIX のファイルがパイプラインなどと同質のストリームを形成するという長所を壊してしまう．

このことは，日本語文字であることをそのバイト列自身が示すようなコード系で文字列をファイルに収めておくという必要があることを示している．コード系の詳細については**3章**に述べる．

UNIX においてCの果たす役割が大きいことは万人が認めることであろう．UNIX の日本語化を考える上でCの日本語化を除外することはできない．言語の中に新しい機能を盛り込むことは，常に以前の利用者（あるいは仕様を決定した機関なり人）との摩擦を避けることはできない．Cに日本語を含めたマルチバイトのコード系の文字処理のための仕様変更を持込むことについては，標準化を審議する内外の機関で本格的に検討されるまでには至っていない．Cの改正案の主な点について**4章**で論ずる．

3. 新しい文字セットの導入

本稿は日本語機能についての論述であるが，コード系を論じる本章においては，コード系の枠組みが重要な論点となる．枠組みは，日本語の文字コードだけでなく韓国語，中国語などほかの言語の文字コードの導入にも同様に利用できるものであり，その点で，日本語固有の部分は極力独立に扱うようにして，議論を国際的な拡張に耐えるものにする必要がある．

UNIX の国際化の第一の要求は，ASCII 文字セットだけが利用できるのではなく，各国で標準的に用いている文字セットを複数セット同時に利用できるように文字の概念を拡張あるいは修正することである．

複数セットを同時に利用する必要性は，従来の UNIX でも感じられていたはずである．たとえば，nroff/troff の清書系は，内部で数学記号・ギリシャ文字などのコードを128以上の数値に割当てている．

これらの文字セットがファイル上でも普遍的に使えるようになっていない原因は，利用している大多数の入出力機器や端末が複数文字セットをサポートせず，また，サポートする機器でも追加文字セットの内容が機器間で統一されていないからである．しかし，日本では英数字と日本文字の両方をサポートするデバイス(以後，慣習に従って漢字端末・漢字プリンタなどと呼ぶことにする）がむしろ主流になってきており，表示される文字も，若干の拡張を除けばおおむね JIS C 6226 にある文字（以後 JIS 漢字と呼ぶ）を用いている．このような統一がすでになされているか，それが期待されているヨーロッパやアジアの国々では，複数文字セットの同時利用が不可欠の機能となってきているのである．

日本およびアジアの国の文字セットは文字の総数が94をはるかに越えているので，それを複数バイトの列として扱うか，文字のビット幅をもっと長くするかしなければならない．前章で論じたように，ファイルシステムとパイプラインは現在の姿のまま保持されることが望ましい．その観点からはストリーム上の文字のコード（以後ファイルコードと呼ぶ）のビット幅を8ビット以外にすることは現 UNIX の全面的改造を要する．一方，4章で論ずるようにプログラムによる処理においてはビット幅を長くした文字コード（以後処理コードと呼ぶ）のほうがよい．ストリーム上ではマルチバイト，処理の上では長ビット幅という2本立てで行うというのが，委員会における結論である．

この方針は入出力機器のコード（端末コード）までも統一することを意味しているわけではない．ファイルコードを現実の機器が採用している端末コードに変換するのが個々のデバイスドライバの役割であることは現状と変らない．

8ビットバイトへの文字セットの追加方法には，ISO に定められた情報交換符号拡張法（ISO 2022-1982）によるものと，日本独自の拡張法であるシフトJIS(系)の拡張法[6]によるものがある．前者は国際的な合意を重視する立場で支持されており，後者は現在のパーソナルコンピュータを中心に広がっている応用との互換性を重視するという立場で支持されている．

この選定において，コードの拡大によって施さなければならない UNIX の既存のプログラムやアルゴリズムの変更の量や面倒さがどの程度で，新しいプログラムやアルゴリズムがどのくらい単純で効率がよいかという技術的な評価も重要な点である．既存のプログラム（コマンド）では，ASCII のデリミタ文字について特別な処理を行い，そのほかのコードの文字については加工を行わないという文字処理が多数を占める．したがって，新しく導入されたコードがそういう特定の文字のコードとバイト単位で衝突しなければ，たいていのプログラムは改造の必要がない．

シフト JIS 系は，JIS 漢字を2バイトに分割し，

ASCII と JIS C 6220 カナ（以後半角カナと呼ぶ）以外の8ビットバイトの空きに漢字の第1バイトを押込んだコードである．元祖シフト JIS コードは JIS 漢字の第2バイトに ASCII のバックスラッシュ文字と衝突するものがあるため，UNIX のプログラムのかなりの改造が必要であり，ファイルコードとして採用は難しい．英数字コードとの衝突しかないように改良したユニバーサル・シフト JIS コード[7]も提案されているが，普及するには至っていない．

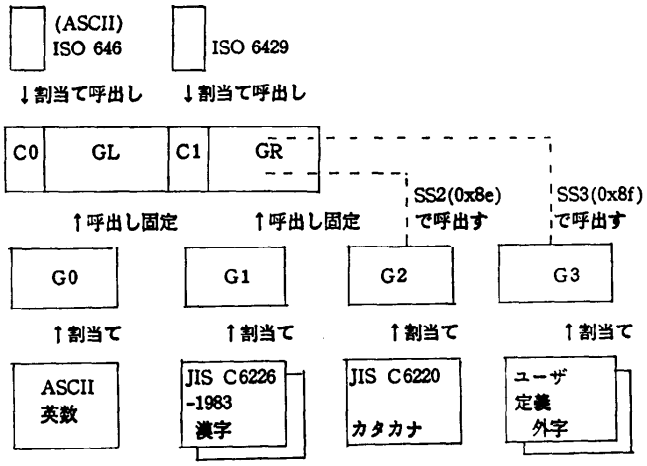

図-1 ISO に則った EUC (JAE 用) のコードセット割当て

ISO 2022 (およびそれと同等の JIS C 6228-1984) は，先行するアナウンス列とコード指示・呼出し列によってさまざまなビット形態の7ないし8ビットのコード空間を形成することができるようになっている．今日まで広く行われているロック型のシフトを使った7ビット2バイトの JIS 漢字のコードもこの拡張法に従ったものであるが，全面的に ASCII と衝突する点でシフト JIS 以上に採用は難しい．

そこで，委員会は同じ拡張法に基づく新しいコード表現を提案した．詳しい説明は省くが，8ビット空間には，常駐の文字セットとしてコントロール文字セット2組と図形文字セット2組，および，シングルシフト文字 (ss2 か ss3) を先行させることによって1文字だけを臨時に割当てることのできる図形文字セット2組を持つことができ，都合4組の図形文字セットを同時に使うようにコード系を設定できる．そして，この割当てによって ASCII だけが符号ビットを0とするような8ビット空間を作ることができる (図-1)．このようにすれば，バイト単位での ASCII との衝突は完全に避けられる．

この割当て方は，アナウンス／指示／呼出し列としてデータ交換の最初に知らせればよい．しかし，これをバイト列の先頭に含めることはストリームに構造を持込むことになる．さいわい，当事者同士の合意のもとで省略できることになっているので，通常の運用においてはそれを省略する．特にそれを含めることをすれば，特別な説明なしで文字セット別のデータの交換を国際的に行うこともできる．

このようにシングル・シフトを使って4つの図形文字セットを同時に使うようにした国際的な体系の文字コードを AT&T は EUC (Extended UNIX Code) と名付けた．JAE では，4つの図形文字セットを順に ASCII, JIS 漢字，半角カナ，ユーザ定義の外字に割当てて使っている．

8ビットの文字コードへ拡張することによる UNIX 独自の問題点は，BYTE の幅が8ビットで ASCII が7ビットであるために空いている符号ビットをほかの目的に利用しているプログラムが存在することである．文字列照合のための特殊文字を処理する sh や ed のようなコマンドは，特殊文字とその文字自身とを区別する目的に符号ビットを利用している．また，TTY ドライバは端末へ出力する文字の符号ビットを直前の文字のための遅延時間を示すのに用いている．これらのプログラムはメモリ効率のためだけにこのような汚いデータ構造を採用しているのであって，国際的な拡張という目的のためにこれを改良することに積極的な反対はなく，AT&T ではすでにそのための全面的改造（たとえば，Korn Shell）が行われていると聞く．

4. C の拡張

プログラミングにおいて，日本文字の必要とされる範囲がどこまでかを決定するのは単純な問題ではない．(1)コメントの中に書ける，(2)処理するデータ型に加える，(3)変数・関数名を日本文字化する，(4)キーワードを日本文字化する，など種々の選択の道がある．(3)は読みやすさを格段に高めるであろう．しかし，入力の手間を考えると，プログラミングの最中に変数名を日本文字で入力するかどうかは疑問

である．(4)は別のプログラミング言語の設計であるとみなすべきであろう．委員会は(1)と(2)を提案し，(3)を積極的に要望することはしなかった．

インプリメントにおいて EUC を前提とすれば，(1)と(3)には技術的な問題はない．以下では，言語仕様の変更をともなうことで問題が生じる(2)について議論しよう．

4.1 long character 型

プログラミング言語でテキストの処理ができるといえるためには，基本的に文字（型データ）の定義・入力・出力・比較・代入が可能でなければならない．さらにこれらの上に文字列の定義・入力・出力・比較・代入が構築され，さらに正規表現パターンとの比較などが必要となってくる．この中でCが文字処理のために構文として用意しているものは文字および文字列の定義と文字の比較および代入のみであり，そのほかは標準ライブラリ関数の形でユーザに提供されている．

現在の言語仕様と標準関数を変えないでマルチバイト文字とシングルバイト文字の混在した文字列を処理するには，シフト JIS 系の基本方針であるマルチバイトの文字をそのままバイト列とみなして処理をするという方法をとらなければならない．その方法は，メッセージ出力のような文字列の定数記述と入出力のみを使う単純な応用でなら複雑な処理なしでうまく機能するが，文字列の個々の文字を取り出して処理する場合は特殊なアルゴリズムを必要とする．

Fortran などと違ってCにおいては文字と文字列は異なる型であり，プログラミングにおける扱いやすさが違う．Cの上でマルチバイト文字を1文字であるにもかかわらず文字列として扱うことにすると，たとえば，switch 文が存在するにもかかわらず，値による多重分岐という文字処理にまさに適合した構文が利用できないというような不都合が生じる．

このような問題はマルチバイト文字の処理だけに固有の問題ではない．たとえば，プログラミング言語のトークン処理は同じ問題を抱えており，「<=」のような2文字の連続はオペレータとして独立した要素とするほうがよい．そのため，前処理で1文字または1整数に圧縮して，それを処理するプログラムを書くという方針で望むのが通常である．

以上の議論に示すように，1文字を1処理単位すなわち基本データ型として扱うということがマルチバイト文字の処理の基本である．日本語の文字を1処理単位とするには現在のところ 16 ビットあれば十分である．実際のビット幅は，将来の都合および計算機に合わせることになるであろうが，とにかく，今までの char 型よりもビット幅が倍以上の新しい文字型をCに導入する必要があることは明らかであろう．

プログラミング言語の拡張を避けるために，上述の処理単位を新しいデータ型として導入するのではなく，整数型の別名 (typedef) にするという方針（JAE リリース 2.0 で供給される日本語化Cはその予定である）はこの場合可能であろうか．Cの char 型の値は1バイト幅の短い整数型の扱いを受けているので，それを単なる整数型としても値の扱いでは差し支えない．しかし，肝心の定数の表記法がないという点で整数型の別名ではうまくいかないのである．

Cの言語仕様を有利になるように読めばマルチバイト文字定数を書けないわけではない．文字定数の値は int 型であり，しかも，シングルクオートの間に複数のバイトを書いてもかまわない（その表す値は処理系定義である）となっている．したがって，クオートされたマルチバイト文字を自然なバイト順に整数に展開した値を結果とするマルチバイト文字定数が書けると主張することも可能である．ただし，この解釈では整数型の上位／下位バイトと文字の第1／第2バイトの対応が処理系定義で，大小関係などの演算のポータビリティが保証されない．

文字列定数を書くには整数配列を割当て（本来なら必要のない変数名を与えなければならない），そこに初期値の形でそれも整数で定数列を与えなければならない．こんな面倒なことを強いたならば，Cでマルチバイト処理を書くプログラマはいなくなるにちがいない．やはり，文字型でなければならないのである．

拡大した文字型を long char 型と名付けることは，Bell 研究所より示唆をうけたものであるが，キーワードを増やさずにすむという利点があるうえに，型の拡張という観点からもまさに的を射た語感の名前といえよう．

この long を使った言語仕様の拡張は，文字および文字列の定数の記法にも適用できる．すなわち，long int 定数を 2L などと書くように後に l または L をつけて

'字' L　"文字列定数" L

のように書くことにするのである．

マルチバイトの文字でグラフィックでないものを表記するために，文字(列)のエスケープシーケンスの書き方も拡張されなくてはならない．シングルバイトと

同様にバックスラッシュの後に8進数(ANSI案では16進数も可)で生の数値を書くことにするのが自然である．文字定数の例にならって l または L をつけてマルチバイトであることを示すには，既存の記法とコンフリクトを起こさないことを保証するために，l をバックスラッシュの直後に置くしかない．さいわい，\l は未使用なので利用が可能である．

以上のように，言語仕様としてはすでにある long という属性を型宣言・定数記述・エスケープシーケンスに適用するという小幅の変更で，拡大された処理コードを導入することができる．

処理コードとファイルコードの対応関係は，あまり複雑な変換を必要としない図-2 に示すものを JAE の日本語化Cが予定しているが，どの文字セットがどういうバイト単位の符号ビットを持つかは本質ではない．

4.2 文字(列)処理関数の拡張

Cのスタンダード・ライブラリ関数に用意されている文字(列)処理関数は long char 型へも拡張しなければならない．拡張には，既存の関数自体の改造と，別名のライブラリ関数の追加の二つの方針がある．Cの言語仕様には引数の型の異なる関数を同名にするオーバローディング機能がないので，引数の型が異なるものを同一の関数で扱うには printf のフォーマット引数のような特別の仕組みが必要である．したがって，文字列を扱う関数は printf 系と scanf 系を除いて別立てとすることが避けられない．そこで，関数の対応だけでも一貫させるため，委員会では関数名の c や s を lc や ls に変えたものとするように提案している．

個々の関数グループについては次のような検討が委員会で行われた．

(1) ストリームを直接には扱わない fopen, fseek などは処理コードのための改造を行う必要はない．

(2) ストリーム以外のファイルコード文字列を扱う system, tmpnam, getenv などは処理コードとの変換を行うものを追加しない．これらは，必要なら(6)の関数を利用する．

(3) long 文字(列)入出力関数は，ファイルコードと処理コードの変換を行う．従来の文字入力関数(getc など)ではマルチバイト文字をバイトに切り離して扱うから，両者を混用することはストリームの解釈に無用の混乱を与えることになる．

	ファイルコード (EUC)	処理コード
制御文字セット 0	000_{xxxxx}	00000000000_{xxxxx} *
制御文字セット 1	100_{xxxxx}	00000000100_{xxxxx} *
図形文字セット 0	$0_{xxxxxxx}$	$000000000_{xxxxxxx}$ *
図形文字セット 1	$1_{xxxxxxx}$ $1_{xxxxxxx}$	$1_{xxxxxxx}1_{xxxxxxx}$ *
または	$1_{xxxxxxx}$	$100000001_{xxxxxxx}$
図形文字セット 2	10001110 $1_{xxxxxxx}$	$000000001_{xxxxxxx}$ *
または	10001110 $1_{xxxxxxx}$ $1_{xxxxxxx}$	$0_{xxxxxxx}1_{xxxxxxx}$
図形文字セット 3	10001111 $1_{xxxxxxx}$ $1_{xxxxxxx}$	$1_{xxxxxxx}0_{xxxxxxx}$ *
または	10001111 $1_{xxxxxxx}$	$100000000_{xxx+xxx}$

* JAE の割当て

図-2 ファイルコードと処理コードの対応

(4) 文字列操作関数 (strcmp など) はその中身がなんであるかは見ていない．したがって，対応するマルチバイト用の関数は同じアルゴリズムでデータの型を long char に変えるだけで作成できる．

(5) lstrlen はシングル・マルチの区別なく文字の総個数を返すという関数にする．strlen で共用していた「入出力する場合のバイト幅」を知る必要があるときには別の関数をユーザが作成する．long 文字列入力関数 fgetls の第2引数も同様に読み込む最大文字数を指定する．文字数かバイト幅かの議論は 4.3 節で述べるが，これらの関数ではすべて文字数とする．

(6) ファイルコード文字列と処理コード文字列の相互の変換関数を新設する．

(7) 文字クラス判定関数，および，文字クラス変換関数 (マクロ定義のものを含む) is_{xxxxx}, to_{xxxxx} は ASCII の文字のクラスを判定・変換する関数である．したがって，(3)，(4)のようなマルチバイト文字一般のための関数ではなく，各文字セットごとに文字クラス判別変換関数を設計するのが自然な考え方である．JIS 漢字の文字クラスとしてどういうものが必要であるかは，現時点では決めないで自然な成熟を待つべきであると筆者は思っている．

(8) ASCIIや半角カナとJIS漢字との相互の変換関数を新設する．引用符のように ASCII 文字が JIS 漢字のいくつかに対応するものがあるため，どういう対応の変換を行うかの合意が必要である．

4.3 書式制御の拡張

書式付き出力関数である printf 系の関数は，タイプライタやラインプリンタのように一定幅の文字をシリアルに出力するデバイスを考慮して設計されたものであるから，スクリーン・ディスプレイのような2次元的デバイスや，タイプセッタのようなプロポーショナルな文字幅のデバイス，さらには両方の特徴を合せ

持つビットマップ・ディスプレイなどには十分な対応力を持たない関数である．漢字端末や漢字プリンタはそのような高度なデバイスであるから，日本文字を導入するのに printf の拡張だけで対処するには大いに無理な面がある．

しかし，printf はそういう高度なデバイスのための（標準）関数の移植性の基礎となるものであり，依然として書式制御は使われ続けるであろう．ここでは，printf の書式制御が本来提供しなければならない機能はなんであるかを考慮しつつ，マルチバイト文字のための拡張を考えていこう．

書式文字列（printf の第1引数）は，単なる引数文字列ではなく，各種の変換を指示するコマンド列であり，ほとんどCの言語仕様の一部であると考えてよい．したがって，その変更・拡張にはCの言語仕様を変えるのと同じ摩擦が予想され，現在の仕様になるべく近い形で，しかも変更幅が小さいことが暗黙の内に要求されている．

書式指定で問題となるのは，マルチバイト文字(列)を表す制御文字をどうするかということと，width の解釈である．さらに，数値変換処理にマルチバイト文字への変換を含めるかどうかという問題もある．

マルチバイト文字(列)であることを示す制御文字として long int を示すのに用いているフラグ l を既存の制御文字 c および s についても用いるように委員会は提案しており，JAEの日本語化Cにも導入される予定である．ただし，UNIX の稼働している機械は 32 ビットのものが中心になってきていて，こういう long 型と int 型が等しい環境ではこの l がなくても正しく動くことが多くなっている．したがって，フラグ l の存在は忘れられつつあるのが現状であろう．存在を忘れかけられたフラグを活用するよりも，必要な s にだけ新しい制御文字（たとえばS）を割当てるほうが簡単でよいとの案もある．しかし，後に述べるように，機能を明確にするためにフラグ方式を推薦する．

第2の問題は，width パラメータの解釈である．この機能は出力の整形（桁合せ）のために設けられているのであるが，非図形文字の印字幅を0にする処理をしてはいない．仮にそうするとしても，端末やプリンタに固有のエスケープシーケンスにまで対処することは現状では困難である．したがって，本心は桁合せを実現したいが，実際は運用上「出力したバイト数」の解釈で行われていると理解すべきである．

マルチバイト文字の導入においても，width パラメータの解釈を実現の簡単な「出力バイト数」とするよう提案している．現在の普通の漢字端末および漢字プリンタの漢字モードの字体の ASCII（英字）と JIS 漢字の文字幅の比は1対2であり，EUC 日本語版のストリーム上のバイト数の比と等しい．そのためこういうデバイスでは英字と JIS 漢字は実質的に桁が合う．しかし，半角カナと外字はシングルシフト文字を持つため印字幅と出力バイト数が等しくない．後2者は使う頻度が少ないとはいえ問題として重大であるから，この程度は対処してほしい．日本で利用する4つのコード単位それぞれの文字幅の比を ASCII＝1, JIS 漢字＝2, 半角カナ＝1, 外字＝2 としておけば，書式による桁合せの不都合を現時点では少なくできると考えられる．

縦横に倍角の字を出したいとか，比が 2：3 のプリンタで桁そろえをしたいというような場合は，printf で間に合わせようとするのではなく，curses のようなスクリーン専用の出力関数ライブラリや troff のようなフォーマッタを漢字端末・漢字プリンタ向きに拡張したものを開発してユーザに提供するような方針にすべきであろう．

3番目の問題は，JIS 漢字で数字や小数点や桁合せのスペースを表示したいとき，書式をどう書くかという問題である．

JIS 漢字のスペースや英数字は一つの文字セット内で閉じた利用を可能にするために設けられたものである．二つの文字セットに同等な文字がある場合は，個個の文字を代表する文字コードはただ一つに限り，重複しているほうはコード空間に存在しても使用しないとする解決法が存在し，個人的にはそれが理想的解決法であると思う．しかし，存在するものを使用しないように徹底させることは至難である．

ASCII と JIS 漢字は現実には半角か全角かで使い分けられているが，字幅を変えることは清書系が本来扱うべき問題である．しかし，出力をすべて清書系にとおすようにすることは現時点では非現実的であろう．

JAE のCは JIS 漢字への数字変換を書式で面倒をみないという方法をとっている．JIS 漢字の数字出力が欲しかったら，いったん sprintf を使って ASCII の文字列に変え，文字列変換関数で JIS 漢字化してから，それを文字列として printf で出力するプログラムを書くことになる．しかし，作業変数を定義しなければならないとか，1ステートメントにきれいに書けないとかがプログラマにとって面倒である．こういう

面倒を避けようとすると，いずれ，そういう変換を行う関数がライブラリとして作られ使われていくようになるであろう．したがって，今のうちに書式を拡張しておいても邪魔にはならないと考える．

委員会はこういう変換をjというフラグで示すように提案している．しかし，こういう変換はJIS漢字に固有であり，国際化の観点からはこの種のフラグをむやみに増やすのは困るという問題があり，その点はjフラグの欠点である．

現在のprintfでは，16進数表記のa〜fや浮動小数点数表記のeを大文字にするのに，制御文字を大文字にすることで対処している．この手法を適用するならばJIS漢字で制御文字を書くようにするという方法になる．制御文字はデータの型を表すとともに出力変形操作を指示するという二重の働きを担っている．さらに，文字セットの選択の機能までしかも多いときは二重に担わせるのは行き過ぎではなかろうか．UNIXの哲学からいうと，それぞれの指定はフラグとして分離されるべきであり，この意味で，JIS漢字表記の導入を機会にこれらを分離してフラグとするとともに，その種類とクラスをうまくまとめるようにするのがエレガントな文法ではないかと考えている．

ライブラリ関数がユーザにとって余分な機能を持つように拡張された場合，それがどの程度までならそのユーザに負担とならないかという問題は，常に未解決である．頻繁に使うprintfにこのような拡張を施した場合，アジアの国のために欧米の国が我慢するだろうかという心配が踏問委員から出されたが，やってみなければ分からない，としか言いようがない．

5. UNIX 上の名前の日本文字化

ファイルコードであるEUCをUNIXの中で使われている名前に使えるようにするのは困難なことではない．同義に扱われているコマンド名とファイル名を始めとしてパスワード・エントリの住所・名前など日本文字化すると分かりやすくなるものはかなりある．

同様にhost名やlogin名も日本文字化できるが，それが必ずしも適切であるとは限らない．host名やlogin名は国際的なネットワークを通じてシステムや個人の識別のために使われるのであるから，国によって変わりうる文字セットは避けるべきであろう．

名前の長さの制限のためにマルチバイト文字を途中で切り離してしまわないように，登録の際にそれを検出して制限することが，名前の日本文字化におけるプログラムの改変の最も大きなものになろう．

日本文字はせいぜい4〜7文字しか名前に使えない．漢語2語ならばほぼ間に合うが，カタカナ語の名前などが入るとこれでは足りないであろう．英語で母音を抜くように「中ピ連」のような省略形がはびこるのであろうか．将来は，BSD版のファイル名のように長さの制限をつけない方向に拡張されることが望ましい．

6. 入力作業の日本文字化

現在の日本文入力処理はほとんどがカナ漢字変換によって行われ，しかも，いわゆる日本語ワープロと切り離せない状況にある．現状の日本語ワープロは機能が単能でなく，単に入力に使うにしては過剰な機能を持ちすぎているが，そのわりにエディタとしては低機能である．日本文入力の導入方式の一つは，編集機能をUNIXふうに汎用化・高度化した日本語ワープロを移植することであろう．そして，現在emacsを使って仕事をするユーザがemacsをシェル代りに使って自分の作業環境を作り上げ，UNIXの環境には戻ってこないのと同様に，日本語ワープロの中だけで仕事をするという形態を目指すことになるであろう．

最近のパーソナルコンピュータではカナ漢字変換部分だけを独立させて入力に利用することが可能になっている．もう一つの日本文入力方式は，このような独立したカナ漢字変換を端末入力に適宜差し入れられるようにすることである．ワードプロセッシングに必要な高度な編集は汎用のエディタ側が受け持つのがUNIXらしい姿といえるであろう．

カナ漢字変換入力機構は，同音語の選択を考えればTTYドライバの位置にあることが本来の姿であるが，辞書アクセスをするにはユーザ・プログラムの位置にいなくてはならない．このジレンマを解くために，daemonの姿にするなどの種々の工夫がなされてきた．System V Rel. 3.0に組み込まれたTTYドライバのストリーム化[8]によってこの問題はたぶん解決されるであろう．ストリームI/Oは，フィルタをデバイス・ドライバの中に組み込んだようなもので，ファイルや共有メモリとのアクセスも可能である．これによって，プログラムの側からカナ漢字変換を呼び出すような今まで実現できなかったことまで可能になる．さらに，変換機構を動的に差し替えたり取り去ったりできるので，種々の変換方式（およびそれ以外の高速入力方式）を個人の好みに従って適宜選択できるようになるはずである．

未解決の問題には，次のようなものがある．

（1） マルチユーザのもとで辞書の個人的チューニングをどうやって行うか．

（2） 差し替え可能なカナ漢字変換のために，どんなデータを端末の属性として用意するか（たとえば，変換キーの種類と呼び名など）．

（3） 差し替えで，辞書などを共通に使うにはどういうインタフェースにするか．

7. メッセージの日本語化

UNIXの日本語化の中でOSやコマンドのメッセージを日本語化したいという要望は第1ランクに位置付けされるであろう．国際化という立場でこの問題を考えると，各国でメッセージを翻訳したソースプログラムを用意するという方式はいかにも非能率である．メッセージをプログラムの中に埋め込むのではなく，ファイルにまとめ，それを環境に合わせて切り替える方式が現実的である．

PDP-11用のBSD版UNIXのコマンドにはプログラムの大きさを小さくするためにソースプログラムからメッセージをファイルに抽出してあるものがある．メッセージの抽出とその出力関数の埋め込みは自動的にmkstrというツールで行うため，プログラミングの手間は増えていない．こういうUNIXらしいエレガントなツールがメッセージの各国語化には必須である．ただ，この方式は後からメッセージの編集ができないとかパラメータを持たせられないなど欠点もあるためそのまま応用できるわけではない．各国語化したときパラメータの語順をどう調整するかとかメッセージファイルの個数や配置の統一などの問題もある．

UNIX全体のメッセージの各国語化は改造の量も桁違いに大きいので実現はかなり困難であろう．自動翻訳の実現によって一気に実現が進むことも予想される．

7. おわりに

本稿で力説したファイルコードと処理コードの2本立ては，論理の道筋として得られた自然な方法である．しかし，言語の拡張の摩擦などを避けるためにシフトJIS系を使ってあくまでバイト列として処理をするのがよいという主張[7]も根強く残っている．これは，マルチバイトの対応のためにアジアの国がUNIXの世界で孤児になることを恐れるからというのが本音であろう．したがって，AT & Tが保証をするUNIXの基準System Vインタフェース定義書[9]に国際化および各国化が定義され，世界のどこで作ったアプリケーションも原則としてどこででも動くことが保証されることが大切である．そのためにも，日本およびアジアの国の大勢が固まる必要がある．

UNIXの日本語化は始まったばかりである．現在は日本語化のためのほんの基本の道具をそろえている状態である．一般に，いろいろな要求が高まり，よい道具がそろって，その上で試行が繰り返されて，よいシステムができ上がる，というのが計算機の世界の発展である．文書処理や作業環境などのUNIXで発展してきた分野が正にそうであり，このUNIXの日本語化によって，現在の日本文ワープロとは異なった高度な新しい日本語処理の発展が期待される．

謝辞 日本語UNIX諮問委員会において討論および資料の提供をいただいた委員長石田晴久教授（東大）および門田次郎氏（AT & T UNIXパシフィック（株））を初めとする委員およびオブザーバの多数の皆さまに感謝致します．

参考文献

1) 大野 豊：Σプロジェクトとは何か，Computer Today, No. 16, pp. 4-11 (1986).
2) 林 秀幸：日本語化が進むUNIX，日経コンピュータ，No. 69, pp. 71-89 (1984).
3) 日本語UNIX諮問委員会：UNIX日本語機能提案書，Press Release，解説“NEレポート，日本語版UNIXの標準案を提案，”日経エレクトロニクス，No. 370, pp. 111-113 (1985).
4) Ritchie, D. M. and Thompson, K.: The UNIX Time-Sharing System, Comm. ACM, Vol. 17, No. 7, pp. 365-375 (1974).
5) Ritchie, D. M.: The Evolution of the UNIX Time Sharing System, Lecture Notes on Computer Science, No. 79, Language Design and Programming Methodology, Springer-Verlag (1980).
6) 長谷川均，岡崎 健：漢字CP/Mのコード体系，情報処理学会マイクロコンピュータ研究会資料，26-2 (1983).
7) 木下 恂：標準プログラム言語における日本語処理，情報処理，Vol. 26, No. 3, pp. 226-232 (1985).
8) Ritchie, D. M.: A Stream Input-Output System, The Bell System Technical Journal, Vol. 63, No. 8, Part 2, pp. 1897-1910 (1984).
9) System V Interface Definition, Volume I and II, AT&T Customer Information Center (1986).

（昭和61年10月1日受付）