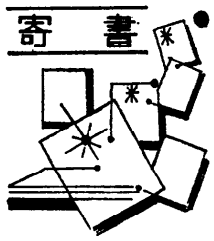

標準プログラム言語における日本語処理†

木　下　　恂††

1. はじめに

標準プログラム言語に日本語処理機能を導入する際の問題点について述べる．これら問題点の多くは，日本語 FORTRAN の言語仕様を統一しようという試みの過程で出てきたものであるが，FORTRAN に限らず他の標準的なプログラム言語の日本語化でも問題になる事柄である．標準プログラム言語における多字種サポートの問題というふうに一般化して考えることもできる．

ここで**標準プログラム言語**とは，国際規格，JIS 規格などで標準化された，あるいは標準化されようとしている言語を指すが，標準化されていなくても de fac'to standard の存在する言語も考慮に入れている．

日本語処理機能とは，正確には日本**字**データを処理する機能のことであるが広く“日本語処理”というふうに使われているのでここではその表現法に従う．以下，**ANK 文字**とは，JIS C 6220[1] による 7 単位または 8 単位符号を，**日本字**とは，JIS C 6226[2] による 2 バイト符号を指すものとする．

2. 日本語プログラム言語の現状

FORTRAN, COBOL などのプログラム言語は，国際規格[3),4)]および JIS 規格[5),6)]により標準化され互換性が保証されているので日本国内でも広く用いられてきた．ところで近年，漢字コードの入力が容易になるにつれ，これら英字をベースにした標準のプログラム言語でも日本語処理を可能にしたいという要求が高まり，日本語 FORTRAN とか日本語 COBOL と呼ばれる処理系が出現してきた．

これらの処理系で共通していることは，標準仕様内で日本語処理を行うのではなく，標準仕様に対し言語仕様上の拡張を行って対処している点である．しかも各メーカごとに拡張の仕方が異なっており，このまま放置するとプログラムの互換性上重大な障害となることが予想され，早急に日本語処理機能に対する国内標準を作るべきであると各方面から提言されるに至った[7)]．

これからのソフトウェア開発環境を考えるとき，特に日本では，その道具の一つとなるプログラム言語で日本語処理ができることは必要不可欠の条件となりつつあり，何らかの統一的な対応が望まれる状況にある．

3. 実現法の現状

（1） 日本語機能の導入形態

プログラム言語に日本語機能を導入する形態には次の 3 つのレベルが考えられる．

① データ処理レベル……言語の中でデータとしてのみ日本字が扱える．プログラムの中の注釈としても使える．

② 識別子レベル……①に加え，データ名や手続き名などの識別子にも日本字が使える．

③ 構文レベル……②に加え，構文も日本語化する．

独自の言語を新たに設計する場合なら③のレベルも考えられるが，国際的な標準のある言語では①のレベルまでが現実的であり，既存の日本語処理系でもこのレベルが多い．COBOL では②まで実現している処理系の例がある．

（2） 言語仕様

FORTRAN については 7 社，COBOL については 11 社の言語仕様を調査した．言語仕様上からみると通常の文字型とは別に**日本語型**を設けている例が多い．日本語型のデータ入出力，日本語型の定数の表記法，および日本語型の演算が定義されている．**表-1** に日本語 FORTRAN における言語仕様拡張の例を示す（詳しくは文献 7), 8)を参照）．各社のコンパイラの言語仕

† The Two Byte Character Processing in Standard Programming Languages by Shun KINOSHITA (TOSHIBA Corp.).

†† (株)東芝青梅工場基本ソフトウェア第 1 部

表-1 日本語 FORTRAN における言語仕様拡張の例

拡張仕様	例
日本語型の宣言	NCHARACTER A*2
日本語型の定数	
● 日本語直接指定	NC '単価'
● カナ漢字変換指定	NN 'ショウ#2'→使用
● ひらがな変換指定	NH 'ハナ' →はな
● 1-2 バイトコード変換指定	NA 'ハナ' →（2バイトコードの）ハナ
● 区点コード指定	ND '42382581' →米国
● 連想コード指定	NK 'ウルウエ' →売上
日本語型の演算	
● 代入	A=NC '単価'
● 比較	IF (A.EQ. NC '売上') I=I+1
● 部分列式，連結式	NC 'アイ'//NC 'ウエオ'
● 入出力	FORMAT (NC '名前', N)

表-2 日本語 COBOL における仕様相違例
(文献 7)より引用)

分類	項目	相違内容
構文の相違	日本語直接指定の定数	NC "…" (3社), ND "…" (1社), "…" (1社)
	カナ漢字変換指定の定数	NN "…" (4社), NI "…" (1社)
	1-2バイトコード変換指定の定数	NA "…" (3社), NK "…" (3社)
	区点指定の定数	ND "…" (1社), J "…" (1社)
	日本語項目	PICTURE N (5社), PICTURE J (1社), PICTURE X (1社)
意味の相違	転記/比較	日本語項目間だけの転記/比較可能(3社), 日本語項目と集団項目間も可能(4社), さらに, 日本語項目と英数字項目間も可能(3社)
	比較	大小比較が可能(3社), 大小比較禁止(4社)
	入出力	漢字シフトコードはレコード名の構造にもとづいて挿入する(3社), さらに, FROM 句の一意名にもとづく(3社)
	報告書作成	日本語の1文字を2カラムとして数える(3社), 1.5 カラムとする (1社), 文字サイズの指定に従う(1社)

様は機能的にはほぼ同じであるが細かい点になると表現法も含めてメーカごとにかなりの差がみられる. 表-2 に COBOL における仕様相違例を示す. 例えば, カナ漢字変換指定の定数の表記法には NN と NI の2種類が存在する. 日本語項目を指定するピクチャ文字にはNとJとXの3種類が存在する等である. 詳しくは文献 7),9)を参照されたい.

(3) 処理方式

処理方式としては, FORTRAN の場合7社を調べた範囲では, コンパイラを拡張して日本語処理機能を追加する方式（5社）と, コンパイラに対し前処理系を設けて標準仕様に変換する方式（2社）とがある.

前者は日本語型という型が一つ増えるので, コンパイラおよび実行時サブルーチン類にかなりの改造が必要となる. 一方, 前処理系による方式では日本語処理機能の文を原則として標準仕様内での記述に変換し標準仕様のコンパイラで処理する方法をとる. したがって実行時サブルーチンは標準仕様のままなので, 特に入出力機能では日本語処理機能に制限を設けねばならない. このように処理方式に依存して日本語処理機能の仕様が決められている場合がある.

COBOL の場合は, コンパイラ改造方式が多いようである.

(4) 日本字コード系

各社の採用している日本字のコード系は, JIS の標準コードまたはそれに適当なコード変換を施したものが多い. 一方, まったく別のコード系を採用しているところもありまちまちである.

4. 規格化する上での問題点

このような既存の日本語処理系の現状を考慮に入れた上で日本語 FORTRAN と日本語 COBOL の統一仕様案が作られた. 日本語 FORTRAN は文献 8)として公表され, その改定版が日本工業標準調査会により暫定規格として建議された. 日本語 COBOL についても近く公表される[9] 予定である. 以下では, この作業過程で経験された日本語処理系の規格化に関する問題点について述べる.

(1) 仕様の永続性と統一性

言語仕様の拡張の仕方にもそれぞれの言語ごとに定石がある. 一般に言語内で扱うべき問題と言語外の問題とは明確に区別されなければならない. 例えば FORTRAN の場合, 日本字の入力手段としてのカナ漢字変換指定, ひらがな変換指定, 連想コード指定などが用意されている例が多いが入力技術の発展にともなってこれらは不要となる可能性のある仕様である. このように長期的視野にたったとき無意味となるような仕様, あるいは特定のハードウェア技術に依存するような仕様は言語仕様に含めるべきではない. また, 前処理系による処理を前提としている場合には言語仕様に統一性がない場合がある. 例えば FORTRAN において, 日本語型データに対する並びによる入力で

は，前処理系を前提にしコンパイラや実行時サブルーチンの拡張をしない方針にするとその実現がむずかしくなる．その結果，並びによる出力はできるが入力は許されないというような不統一な仕様になってしまっている．

日本語 FORTRAN の統一仕様案作成に当っては永続性のある仕様に限定し，前処理系を前提としないで言語仕様を設定する方針がとられた．

（2）互換性

すでに使われている日本語 FORTRAN や日本語 COBOL では細かい仕様を比べるとメーカごとに仕様が異なり互換性がないことはすでに述べた．これらの仕様を統一し標準化しようとしたが，文献 8)，9)にみられるように各メーカの利害がからんできて統一は困難を極めた．一応統一案は完成したが処理系定義として各処理系に仕様決定をまかせた部分がすくなくない．特に ANK 文字1字と日本字1字との間には何の関係もないとする考え方が支配的であるが，対象にしている言語のレベルではこれを無関係として全体的つじつまを合わせるのは難しいことがしだいに明らかになってきた．詳しくは後述する．

（3）シフトコードの扱い

日本語処理機能を考えるときシフトコードの扱いが最大の問題点となっている．即ち，日本字のコード[2]は，ANK 文字2個の組合せで表現されるので ANK 文字のコードと重複があり，コードだけで ANK 文字と日本字とを判別することはできない．そこで ANK 文字と日本字との間にエスケープシーケンスを挿入して区別する方法がとられている．このエスケープシーケンスにも標準[10]があり，通常**図-1** のように前後に3文字から成るエスケープシーケンスを挿入する約束になっている．このためエスケープシーケンスの分だけファイル量が増大する欠点があり通常もっと短いコード，例えば**表-3** のようにして別のコードをエスケープシーケンスの代わりとして用いている場合が多い．通常このコードを**シフトコード**と呼んでいる．

シフトコードは表-3 のように各社各様であり機種に依存すると考えてよい．したがってそれを含むファイルは異機種間で互換性がない．一般に ANK 文字と日本字の境界を示すシフトコードの操作を言語利用者にまかせてプログラム言語で直接取り扱うようにすると，利用者にとって非常な負担となる．それは文字列を表示したとき実際には表示されない(見えない)文字を通常の文字と同じに扱わねばならず，しかもシフトコードはファームウェア等で自動的に挿入されるものであり利用者が意識して挿入したものではないからである．シフトコードを直接扱うときの問題点を**表-4** に列挙する．これらの問題点を含むファイルを扱うアルゴリズム自体に互換性がなくなるのは明白である．

6ビット表現の BCD コードが主流の時代にもカナ文字を導入するために，これと同じようにシフトコードを用いた．利用者は目に見えないシフトコードの分まで字数に入れて数えなければならなかったので非常に使いにくかった．コンパイラ作成者にとってこのときの苦い経験から ANK 文字への日本字導入ではこのシフトコードをできるだけ言語利用者に見せない，あるいは意識させない方式を選ぶことにしたのはそのためである．即ち，文字型と日本語型とをはっきり分けて混在することがないようにしたのである．こうすると文字型と日本語型とを区切る情報(シフトコード)は必要がなくなるというわけである．しかしファイル

表-3　各種のエスケープシーケンスの例

実施例	バイト数	日本字の呼出し	ANK 文字の呼出し
JIS 標準	6	ESC*, 2/4, 4/0	ESC, 2/8, 4/10
A 社	6	ESC, 2/4, 4/0	ESC, 2/8, 4/8
B 社	4	1/10, 7/0	1/10, 7/1
C 社	4	0/10, 4/2	0/10, 4/1
D 社	2	2/8, または 3/8	2/9
E 社	4	1/14, 12/1	1/14, 12/0
F, G社	6	ESC, 2/4, 4/0	ESC, 2/8, 4/10
F 社	3	ESC, 8/1 *n*（*n* で日本字のバイト数を示す）	な し

* ESC のコード値は，列番号/行番号で表すと 1/11 である

表-4　シフトコードの問題点

(1)	表現形式が不統一
(2)	長さが不統一
(3)	挿入場所が不統一（前／後または前だけ）
(4)	シフトコードを含む文字列のままでは大小比較ができない
(5)	行頭（または改行の直後）では常に ANK モードとしているファイルがある
(6)	シフトコードを含むファイルは互換性がない

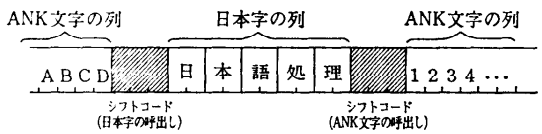

図-1　エスケープシーケンスの使い方

への入出力を扱う段になるとシフトコードの存在を完全に無視することはできなくなり，頭かくして尻かくさずという格好になっているのが現状である[8),9)]．それは，ここで対象としているプログラム言語の多くが（書式付き入出力等で明白に変換している場合を除いて）メモリ上のデータ表現とファイル上のデータ表現とは同一であるという暗黙の了解の上に言語仕様を決めているからである．書式付き出力で変換する場合でもANK文字と日本字とが混在して出力されると桁とか**記録の長さ**に言及するところで破綻をきたす場合が多い．

（4） 操 作 性

日本語型を導入した日本語FORTRANや日本語COBOLの仕様には機能上大きな欠陥がある．それはシフトコードを利用者に見せないという基本的な設計方針にもとづいて文字型と日本語型を分けてしまったためANK文字と日本字の混在したデータ（以後**混在データ**と呼ぶ）を扱うことができない点である．ファイル上でANK文字のある欄と日本字のある欄が利用者にとってあらかじめ分っている場合は利用者の責任で入力可能であるが，欄が確定していない場合には，型の属性が異なるため入力することはできない．例えば，日本語FORTRANによるプログラムテキスト自身を考えてみよう．ANK文字から成るプログラムの中に日本語定数や日本語注釈などが散在しているわけであるから，このようなデータを日本語FORTRANで扱うことはできない．プログラムテキストのような混在データはFORTRANやCOBOLでは扱う必要はないという主張もあるが，AdaやCあるいはシステム記述言語などではプログラムテキストをデータとして扱わねばならないのでこのような仕様は一般的ではなく採用しにくい．

（5） 改 造 量

コンパイラの改造によって日本語処理機能を導入する場合，日本語型という新しい型を一つ導入するための影響がコンパイラ全体に波及し改造量は予想以上に大きくなる．例えばFORTRANの場合外部仕様を見ただけでも以下のような文に仕様上の変更または仕様の拡張が必要となる．

- 宣言文
  - EQUIVALENCE文
  - COMMON文
  - NCHARACTER文(☆)
  - IMPLICIT文
  - PARAMETER文
- DATA文
- 日本語代入文(☆)
- 入出力文
  - OPEN文
  - INQUIRE文
- FORMAT文
- FUNCTION文
- 組込み関数

（☆）印は新しく導入された文

その他実行時サブルーチンの改造も必要である．

一方，前処理系を用いて日本語処理機能を導入する場合は標準のコンパイラはいじらないでよいので日本語処理機能を必要としない利用者には迷惑がかからないという利点がある．しかし次のような欠点もある．

① 前処理系を通すためのフェーズが増えるので全体のスループットが落ちる．

② コンパイラのエラーメッセージと前処理系が扱うソースのリスト上での対応がとりにくい．

また，日本語機能を実現するために内部的には2バイト符号1文字を2つのANK文字とみなすためコンパイラのエラーチェックをはぶくとか多少の手直しをしているのが普通である．

このような日本語処理機能独特の改造を各種のプログラム言語で等しく行わなければならないとすると，全体として膨大な投資が必要となる．ソフトウェア危機ということがいわれて久しいが，日本のみがこのような投資を今後とも永続的に強いられるとすれば，諸外国に比べ大きなハンディキャップをしょいこむことになる．プログラム言語の種類に依存しないような何らかの日本語処理機能のための共通技術の確立が必要である．

（6） 規 格 化

日本語処理機能の仕様を統一し日本工業規格として標準化し，さらに国際機関に提案すべきであるという考え方がある[7)]．独自言語を設計するのであれば何ら問題はないが，すでに国際規格のある言語に対し新しい機能拡張を行ってJIS化するということは，手続き的には言葉で言うほど容易なことではない．一般に標準仕様に対し**処理系ごと**に仕様を拡張することは，一定の約束のもとに許されている．しかし**国ごと**に標準を変えるということは，特別な場合（通貨記号の表現など）以外許されていないのが普通である．したがって日本のみが特別であって日本語仕様を含む新標準を設定してよいという議論は通用しにくい．各国がそ

のような行動に出たら国際標準はまったく意味をなさないことになってしまう．また，互換性というものは相互に可能なものでなくてはならず，海外の標準的なプログラムは日本で利用できるが日本語プログラム*は海外では利用できないというような一方通行が許されるはずがない．そのような特別な規格に合致していないと日本では標準とはいえないとするならば，それは非関税障壁の一つとみなされて貿易摩擦の原因となる可能性すらある．

幸い日本語 FORTRAN の統一仕様案は JIS 規格原案として扱われることになり，当面の仕様上の混乱は回避できそうである．しかし，これを本規格とするにはさらに広範囲に意見を求めて仕様の洗練化をはかる必要がある．また国際的に認知させるには単に日本語処理機能を指向するのではなく，特定の国に依存しない多字種サポートとして仕様を一般化させる必要があろう．

5. 日本語処理機能の実現法

5.1 標準仕様内での対応

言語仕様を拡張して日本語処理機能を実現する方式では 4. で述べた問題点のうち(3)以外のものを避けることはできない．これらの問題を根本的に解決するためには，日本語処理機能を標準仕様内で実現することを考える必要がある．FORTRAN は英字をベースにした言語であるが英語のための機能はない．同様に，日本語 FORTRAN も日本字が扱えればよいわけで日本語を扱うための特別な機能は不要である．日本字を扱うということは文字通りコード系の問題であってコード系さえ工夫すれば自然の形で標準仕様内で日本字が取り扱えるはずである．従来，シフトコードの問題があるため標準仕様内での対応を断念してきたが，その裏にはコード系をいじることをタブー視し，どのような日本字コードを採用していても実現上問題のない方式を追求してきたという背景もある．しかし最近のプログラム言語では高度の互換性を保つためにはコード系まで含めて規格化しなければならないという認識になってきている．その意味では日本字コードも聖域ではなくなってきているといえる．

5.2 コード変換の必要性

日本字を ANK 文字の自然な形での拡張と考えることによって混在データを扱えるようにしたい．そのためには，いわゆるシフトコードを用いないでも ANK 文字と日本字の判別がつくようなコード体系に変換（マッピング）する必要がある．

マッピングが必要となるもう一つの理由として，次のような問題がある．通常，文字列定数を表現するのに文字の列を二重引用符（"，コード値を列番号／行番号で表示すると 02/2），または単引用符（'，02/7）で囲んで表す言語が多いが，文字列の中に " または ' 自身の値に相当するコードを書きたいときは利用者の責任でそのコードを2つ並べて書き，1つの " または ' に対応させるという規則が一般的に使われている．文字列の中に日本字を書く場合，JIS 日本字コード表の2区または2点に属する文字は " と同じコードを含んでおり，7区または7点に属する文字は ' と同じコードを含んでいる．その結果これらの日本字を文字列定数の中で用いる場合に限り，標準仕様の規則どおり，利用者の責任で " または ' を重複させなければならないことになる．これは利用者にとって大きな負担である．しかも，日本字コードの間にシフトコードの挿入なしで " や ' を入れることはできないので通常の手段ではこれは不可能な要求である．JIS C 6226 の日本字コードは，当然のことながら，このような言語側の事情はまったく考慮せずに定められているので，JIS の日本字コードそのままで日本字を扱うことは，これまでの言語仕様に従う限りできないことがわかる．そこで既存の処理系では通常何らかのコードの変換を行い，" や ' のコードを含まないコード系にマッピングして用いているものが多い．

プログラム言語の規格では処理系ごとに処理系定義の文字を追加してよいことになっている*．この点に着目し8単位符号の ANK 文字の未使用部分に日本字用の特殊文字を追加し，そのようなコードの組合せで日本字コードを表現することにする．これによって約 8800 字もの日本字の字種を追加する代わりに，高々数十個の字種の追加で日本字サポート，あるいは多字種サポートが可能となる．

5.3 日本字コードのマッピング法

日本字コードのマッピング法としては以下のようなものが提案されており，一部は実際に使われている．

(1) シフト JIS 漢字コード

国際的には文字セットとして通常7単位符号が使われているが，日本ではカナ符号を含めて8単位符号が

* 日本語特有のプログラムは，海外で利用されることは実際上ないかもしれないが，単にデータとしてのみ日本字を扱えるようにした一般的なプログラムなら海外で使えるであろう．

* ただし，Ada は例外である．

主流である．シフト JIS 漢字コード[11]は，8単位符号のカナ文字を生かし，それ以外の未定義の部分08列，09列，14列，及び15列の64か所を用いる．JIS 日本字コード表は第一，第二水準までで94区×94点なので最低94個の空きが必要となりこのままでは収容しきれない．そこで $94\times94=(94/2)\times(94+94)$ と考えて 47(=94/2) 個の空きを隣り合う奇数区 ($2\times n-1$ 区，$n=1$〜47)，偶数区 ($2\times n$ 区) のコードとして意味付け，そのコードを日本字の第一バイトとして用いる．第二バイト目に用いるコードとしては，その奇数区の第1点から第94点まで，及び偶数区の第1点から第94点までを表わすコードとして188個の文字を選ぶ必要があるが，これは"や'のコードと重複しない限り他のコードと重複してもよいことにし，@ (04/0) 以下を割り振る．このとき"や'ばかりでなく (.), *, +, ', −, ·, ／などの特殊文字も含まないようにしておくとコンパイラの処理上都合がよい．このような観点でコードの変換を行ったのが**シフト JIS 漢字コード**である．

シフト JIS 漢字コードの採用により ANK 文字と日本字の混在表現が可能になった．しかし日本字文字列の操作に関しては問題が残っている．即ち，①漢字の2バイト目のコードとして [¥] ^ ー ` { | } 〜などの特殊文字を含んでいるためこれらの特殊文字を特別な意味に用いている言語（C言語や UNIX のシェル言語など）では混在データの操作上問題が起こる．また，② “ずれ読み” を起こす欠点もある．ずれ読みとは，2バイトから成る日本字コードの列を1バイトずらして読むことによって隣り合う日本字コードの第2バイトと第1バイトを組み合わせてまったく別の日本字コードと見誤る現象をいう．例えば，……東寺……という文字を含む文字列に対し “月” という文字を検索すると “東” の第2バイトと “寺” の第1バイトが組になって “月” という文字が検出されてしまう．

もう一つの欠点としては，③文字列を構成するバイト列上で任意のバイト位置を与えられたとき，そこに置かれている値を見ただけで日本字なのか ANK 文字なのか，日本字ならその第1バイトなのか第2バイトなのかの判定がつかない点である．最悪の場合にはバイト列を逆向きに日本字用特殊文字でない文字値が出てくるまで走査し，そこから逆算して判定しなければならない．

（2） JIS 8単位漢字コード

情報交換用符号の拡張法[10]に対する新しい規格[12]を適用し，JIS C 6226 漢字コードを8単位符号の 08列〜15列へマッピングするコード系が考えられる．JIS C 6226 の漢字コードに16進で8080を OR したコードと思えばよい．この方法によれば ANK 文字との重複はないので①の問題は起こらない．しかし②のずれ読みの問題はさらに深刻になる．即ち，シフト JIS では，ずれ読みの起こる可能性のある領域は全体の4分の1の領域に属する文字であるが，JIS 8単位符号では全文字ずれ読みの対象となりうる．また，③の問題も逆走査して文字列の頭か ANK 文字が出てくるまで判定がつかなくなる．さらにカナ符号を扱うためにはエスケープシーケンスで呼び出さねばならない．

（3） シフト JIS 汎用漢字コード

上記①，②，③の問題を解決した汎用性のある漢字コードとしてシフト JIS 汎用漢字コード[13]がある．

JIS C 6226 による区点コードとの対応は次のとおりである．

区コード：10/1〜15/14⟶(区 1〜区 94)
点コード：03/0〜03/ 9⟶(点 1〜点 10)
04/1〜05/10⟶(点 11〜点 36)
06/1〜07/10⟶(点 37〜点 62)
08/0〜09/15⟶(点 63〜点 94)

さらにカナ文字の表示を容易にするために8単位カナ符号を2バイト符号化し半角のカナの扱いにしている．対応する領域は第1バイトは区コードとして 10/0，第2バイトは上記点コード領域を利用する．

このコード系の利点は，まず日本字の第2バイトを構成する文字がすべて数字，英大文字，英小文字及び 08〜09 列のコードのみから成り空白を含めて以下の特殊文字をまったく含んでいない点である．したがって，①の問題は起こらない．

[¥] ^ ー ` { | } 〜

次に，日本字コードの第1バイトと第2バイトの間にはコードの重複がまったくないので日本字文字列の検索に際して②のずれ読みを起こすおそれがない．

また，任意のバイト位置を与えられたとき，そこに置かれている値から日本字か否か，日本字なら第1バイトか第2バイトかの判定が容易につくという特長をもっている．最悪の場合でも直前の1文字を見ればよい．

5.4 日本字の扱い

前節で述べた各種の日本字コードを内部処理的にどのように扱うかについて述べる．

（1） 例えば Ada では，“=”と“>”とをこの順に直接並べて“=>”として右矢印を表現している．ここで個々の文字“=”や“>”には右という意味も矢印の意味もないことはいうまでもない．“=>”はコンパイラの内部処理の上から見るとどのような記号であってもかまわないが，リスティング上や CRT 等の表示装置上では利用者に**右矢印**というイメージを想起させるという役割をもっている．これと同じように日本字も内部処理では単に二つの特殊文字（正確には2バイト目は英数字を含む場合があるが，ここではひっくるめて**日本字用特殊文字**と呼ぶことにする）が直接隣り合っているだけの存在とみなし，表示装置上に表示されたとき初めて漢字のイメージになって利用者に読み取られるものと考える．

（2） 日本字は日本字用特殊文字が二つ直接隣り合ったものであるから，言語仕様上で日本字が許される場所はそのような特殊文字の書ける場所に限られる．即ち，データ以外ではプログラムテキスト中の文字列定数の中，及び注釈の中だけである．識別子の中に書きたければ識別子として単に特殊文字を許すように言語仕様を拡張すればよい．

（3） 言語仕様上では日本字1字は（ANK 文字で）2文字と数える．即ち，文字列の長さは常にバイト数と一致すると考えてよい．この結果，"漢 字"という文字列中の空白が漢字空白1つなのか ANK 文字の空白2つなのかにかかわらず文字列の長さは常に6と視認できる．一方，日本字を ANK 文字とともに字づらで1字と数える方式では，この文字列の長さは3なのか4なのか判断できない．

（4） 第1バイトが日本字用特殊文字であるのに第2バイトが日本字用特殊文字でないような文字列を表示しようとしたときの表示結果については保証しない．

6. おわりに

本稿では，日本語処理系の規格化に当っての問題点と，実現法について2つの方式があることを紹介した．日本語処理機能を標準仕様内で実現する方式が広く受け入れられるようになれば，プログラムの互換性上望ましいばかりでなく，日本語 FORTRAN や日本語 COBOL などの日本語処理系の開発に要する膨大な費用を節約することが可能となり，その余力を日本語文法特有のソフトウェア開発に注力できるようになるのではあるまいか．

なお，日本語 FORTRAN，日本語 COBOL などの既存の処理系の仕様についての情報は，日本電子工業振興協会の言語標準化専門委員会の場で，沖，電電公社，東芝，日電，日立，富士通，及び三菱各社の言語開発担当者から得たものである．記して感謝の意を表する．

参 考 文 献

1) 日本工業規格：情報交換用符号 JIS C 6220
2) 日本工業規格：情報交換用漢字符号系 JIS C 6226
3) ISO： 5139-1980 Programming language FORTRAN
4) ISO：1989 Programming Language COBOL
5) 日本工業規格：電子計算機プログラム言語 FORTRAN JIS C 6201
6) 日本工業規格：電子計算機プログラム言語 COBOL JIS C 6205
7) 日本語処理技術に関する調査研究 電子協 58-C-454
8) プログラム用言語の標準化に関する調査 —日本語 FORTRAN—（言語標準化専門委員会報告書）電子協 59-C-483
9) プログラム用言語の標準化に関する調査 —日本語 COBOL—（言語標準化専門委員会報告書）電子協 60年度（予定）．
10) 日本工業規格：情報交換用符号の拡張法 JIS C 6228
11) ASC II, Vol. 7, No. 5, pp. 226-231 (May 1983).
12) 日本工業規格：情報交換用符号の拡張法 JIS C 6228 (1984).
13) Ada の適用性評価（3）—新しい日本字コード系による日本語処理の実現，情報処理学会第30回全国大会論文集 (1985).

（昭和59年10月17日受付）