解　説

2. プログラミング・パラダイムと環境

2.3 日本語プログラミング環境†

高　橋　延　匡††

1. はじめに

プログラムを自分が体得している自然言語で書けたならばきっともっといろいろな問題の解決が容易になるのではないかという考えは自然なものである.

第一世代のコンピュータは，機械語のプログラミングで始まったが，1949年，世界最初のプログラム内蔵方式のコンピュータ EDSAC[1]では基本アセンブラの原型に相当するイニシャルインプットルーチンが開発された．とはいえ，アセンブリ言語によるプログラミングは緻密な作業である．名人芸的なプログラマを除き，一般にはプログラムの生産性も低い．このような状況を一挙に打ち破ったプログラミングにおける最初のブレークスルーは大型科学用コンピュータ IBM 704 を対象に 1957 年に開発された FORTRAN-I[2]であった．"プログラミング言語の発明" である．

この FORTRAN の開発は，プログラミングに大きな影響を与えた．理学部や工学部の学生たちにとっては，自分たちの研究課題を自分でプログラミングできるからである．これを境にして，UNIVAC と IBM の市場評価の逆転が始まった．まさに，市場の要求（容易にプログラムできること）を制するものは市場を制したのである.

FORTRAN に刺激され，自然言語（英語）でプログラミングをしたいという願望の最初の試みは，事務処理に始まった．IBM では，事務処理用言語として COMTRAN を開発したが，これに対し，米国の IBM を除くメーカが米国国防省と連合を組み，COBOL[3] を開発した．COBOL の目的とするところは自然言語によるプログラミングである．確かに，制約条件の下ではあるが，英語の構文規則にのっとり，識別子も 30 字ということも含めて事務処理計算の分野に着実に普及していった．その結果，第二世代，第三世代をとおして，企業活動の中心に COBOL とメインフレームの組合せがある．と同時に，"計算機言語の標準化" という概念を定着させた.

この間，わが国のコンピュータメーカは，IBM にいかにはやく追いつくか，という競争の時代であった．そのような状況では日本語情報処理は，コストがかかるため，新聞社などの特定の分野でしか実用化されなかった[4),5)].

このような状況を大きく変化させることになったのは，1978 年の "仮名漢字変換を用いた日本語ワードプロセッサの商品化" をあげることができる．東芝が開発した JW-10 の誕生である．技術的には，仮名漢字変換だけを採りあげれば，九州大学をはじめとして研究が行われていたし，共同通信社では外電処理にローマ字からの仮名漢字変換を開発し実用に供していた．しかし，この時点までメインフレームメーカは日本語情報処理に対して熱心ではなかった.

1980 年代に入り，LSI 技術の進歩とともに CPU チップが高性能化したことから，仕事に使えるパーソナルコンピュータが誕生した．一方，ミニコンのエンハンス版に位置づけられる高性能ワークステーションが誕生した．前者の OS は MS-DOS が，後者は UNIX が主流になっていることは周知のとおりである.

日本では，日本電気の PC-9801 が市場で大きな成功を収めている．この利用目的で一番大きなものは，日本語ワードプロセッサとしてであることは，ワープロソフトの販売数から推測できる．このことは，日本語プログラミング環境の必要性の潜在的な証明である.

一方，ワークステーションについては UNIX が大きな成功をもたらした．UNIX は，コンピュータユーティリティという概念で誕生した汎用大型 TSS であ

† A programming Environment in the Japanese Language by Nobumasa TAKAHASHI (Department of Information Science, Faculty of Technology, Tokyo University of Agriculture and Technology).

†† 東京農工大学工学部数理情報工学科

る MULTICS の反省として誕生した．多重アクセスからシングルユーザによるアクセスへの方向転換であった（なお，その後 UNIX も多重アクセスが可能になっている）．この UNIX がもたらした，プログラミングに関するブレークスルーは“プログラムの開発環境”という概念の実現である．これはプログラマに大きくアピールするところとなった．この概念の重要性を認識した企業が，市場を制する．さらに，わが国では，次の段階として，“日本語プログラミング環境”こそ，日本の将来の市場を創出するものと考えられる．

2. 日本語プログラミング環境の枠組

ここでは，筆者がもつ日本語プログラミング環境のビジョンとその実現のための設計指針[6]について述べる．

すなわち，次の5つの問題が対象となる．

（1） プログラムの生産におけるすべての過程で，日本語を自然な形で使えること．

（2） これを，一貫したアーキテクチャとして実現すること．

（3） 日本語を扱う，それぞれのインタフェースの正規化をはかること．

（4） 上記のアーキテクチャと既存のアーキテクチャとの間の情報の変換系を開発すること．

（5） 日本語辞書をはじめとする，「日本語のための各種の辞書」を構築し，それを利用できること．

これらは，それぞれ本質的な問題と技術的な問題を含んでいる．以下，上記についてその研究課題を考察する．

2.1 プログラムのライフサイクルをとおして日本語を自然な形で使えること

プログラムのライフサイクルの全工程で，日本語を導入すること．

（1） 要求仕様の作成

要求仕様の作成は，現在，ほとんど日本語で行われている．しかし，その要求仕様から，設計仕様への変換，さらに，検証系への変換など，いろいろの問題が未解決である．これらは本質的な問題である．現実的なアプローチとして，これらの問題に着手するまえに，文書の世代管理が簡単に，かつ，安価に実現できることが望ましい．これはソフトウェアの開発環境の改善になる．技術的問題としては一回だけ書け，あとは読み出し専用になる大容量で安価な光ディスクが一つの解を与えてくれると思う．

（2） 設計仕様の作成

設計仕様の作成も，要求仕様の作成とほぼ同様な問題をかかえている．すでに類似のプログラムがあれば，それを流用するという UNIX 流の現実的な手段がある．設計仕様レベルで，日本語を利用できることから，ソフトウェアの再利用などが容易になるだろう．設計仕様から，プログラムへの変換系や，設計仕様とプログラムとの参照関係を明示するソフトウェアツールを開発することにより，ソフトウェアの生産性，信頼性が高められる．

（3） プログラミング

すべてのプログラミング言語の中で，日本語が自然に扱えるようにする．たとえば，識別子に日本語が使用できることは当然でなければならない．特に，従来のアセンブリ言語や汎用言語ばかりでなく，C, LISP, Prolog などいろいろの言語で，日本語が利用できることが大切である．

これにより，単に，コメント文に日本語が利用できるだけでなく，プログラム自体が日本語を利用できるために可読性が向上し，保守性も大幅に向上すると期待できる．また，設計仕様で代表される文書とプログラムとの一貫性を保証するような緊密な世代管理は，ソフトウェアの生産性向上だけでなく，ソフトウェアの機能拡張の際にも，ソフトウェアの寿命をも向上されることが期待できる．

（4） テスティング，保守

日本語プログラミング環境は，要求仕様，設計仕様から並列にテスト仕様を生成する研究を実務レベルで可能にする．また，システムの改版（バージョン・アップ）時に，テスト仕様に基づく組織的なテストがやりやすくなる．保守性の向上は言うまでもない．

（5） 文 書 化

ソフトウェアの開発過程で，日本人にとってもっとも苦手とされ，しかし，絶対必要なものに文書化がある．文書化がうまくいかない原因の第一は，日本の国語教育や，大学の工学部における「テクニカルライティング」の教育がないという指摘がある．

第二に，日本の伝統文化からの影響で“沈黙は金”というように，言いたいことは腹に収め，あまり表現したがらないことによる．

第三に，日本の企業は転職というのが例外という原則で，成り立っていることによる．したがって，文書化で仕事が行われるというよりは，人がデータベースになって仕事が行われているという面があることで

ある．

また，プログラム開発現場をみると，日本語の計算機処理が貧弱だったこともその一つである．たとえば，10年前には非合理的な精神主義で，薄い紙に鉛筆で仕様を書かせ，青焼した文書を配るというような状況にあった．この点に関しては，日本語ワードプロセッサ（ワープロ）が開発され，手軽に利用できるようになった結果，大幅に改善されつつある．しかし，現状ではワープロの文書のファイルはフロッピィディスクで管理されている．これは，上記の文書も含んだトータル・システムを支えるには不十分である．しかし，入れものの問題は，技術的な問題であり，磁気ディスクをはじめ光ディスクなどで大幅に改善されると確信している．

むしろ，本質的な問題として，プログラミングの過程で作成される文書の計算機による理解の問題が最大の挑戦すべき課題である．

なお，これらは米国における UNIX 流プログラムが一つの典型である．日本で単にローマ字に置き換えたものでは，多くの人たちの満足するものにはならないことは自明である．

2.2 日本語プログラミング環境のための一貫したアーキテクチャを実現すること

日本語プログラミング環境を実現するためには，まずコード系を確立することである．基本的には，文字の形を表す属性情報と，文字のコード情報とは厳密に区別することが基本である．もちろん，その他にも図や表の処理も容易でなければならない．これらが，ファイル・システムも含めて一貫したアーキテクチャで取り扱えることが必要である．

また，このように，一貫したコード系を基礎に置くアーキテクチャでは，実現される言語処理系が，共通したコード系をもつことができる．かつ，プログラミング言語間で，実行環境の統一をはかることにより，ユーザの使い勝手の向上がはかれる．

次に，入出力に関しても，基本的に2バイトコードをもつテキストファイルと，文字情報をもつ属性ファイルとを利用することにより，多様な入出力に対応可能となる．

2.3 インタフェースの正規化をはかること

日本語プログラミング環境の構築には，入出力装置の接続をはじめとして，インタフェースの問題がある．ここであえてインタフェースの標準化と言わずに正規化という意図は，次の理由による．インタフェースに関しては，標準化の努力がなされている．しかし，JIS の標準化にしても，すべてが理想的なものではない．むしろ，現実と理想との妥協の産物として制定されたものがあるのは当然である．また JIS 化には時間がかかるため，現実には製品の方が先行してしまうこともまれではない．ここで言う正規化とは必ずしも従来の互換性に捕われることなく，計算機システムを構成するいろいろのレイヤ（OS, 入出力，ユーザなど）間で，理論的見地から理想と考えた約束ごとを決めることと定義する．したがって正規化と標準化とが近いほど望ましいと考える．

以下，文字コードについては **3.3** で述べるので，それ以外について，典型的な例を考察する．

（1） キーボード

入力については，キーボード一つを取りあげてみても，いくつかの方式がある．むしろ，キー配列の規格を作るよりは，キーボードとコンピュータとのインタフェースの標準化が必要である．それによって，個人の好みにあった配列のキーボードを利用することが可能となる．このようなインタフェースをもつキーボードの実現方法としては，キーボードの中にインタフェースの仕様を実現するためにマイクロプロセッサを埋め込む方法や，OS の機能としてその処理プログラムを用意する方法がある．現に，ワープロの仮名漢字変換用のキーボードの新配列は JIS 化されたが，市場ではほとんど無視されている．この原因の一つには，互換性を意識した結果，理論的見地から最良のものを作るという立場を採用できなかったこともあると思う．

（2） ワープロ文書ファイル

ワープロ文書ファイルは各社がまちまちで，かつ非公開である．そこで，8インチのフロッピィディスクを用いた文書交換のための変換用フロッピの JIS 化（JIS X 4001）が事務機器工業会を中心に行われた．この段階で，正規化に基づく標準化の検討が十分に行われるべきであったが，すでに各社が独自の規格で製品を出していたため，各社の部分集合的な規格となってしまった．このため，ワープロの図形出力部分の JIS 化はなされていない．その結果，互換性に問題が生じ，事実上ほとんど意味のない標準化になってしまっている．

以上，正規化の視点に欠けた標準化を例示した．われわれは，標準化をする際には，インタフェースの正規化の研究と努力が必要と考える．

2.4 アーキテクチャ間の情報の変換系を開発すること

現実には，既存の各種の OS がある．それらを使用して，すでに作成したプログラムや文書ファイルを，新しい日本語プログラミング環境のアーキテクチャに移すための各種変換系が必要となる．また，この逆も同様に必要である．このような変換系の実現を容易にするためにも，前述のインタフェースの正規化が必要となる．

われわれの経験によれば，日本語ワードプロセッサをオフラインのプログラマーズ・ワークベンチ (PWB) として使用[7] したが，この変換系のソフトウェアは大変有効であった．

2.5 日本語辞書整備と日本語処理のための各種の知見の構築をはかること

日本語プログラミング環境を構築する際の本質的問題は，日本語自身の機械処理の研究とノウハウの構築がまだ十分でない点にある．たとえば，国語辞典を単に計算機に入力しても，決して情報処理のために良い辞書にはならない．人間のための辞書と機械処理のための辞書とでは，おのずからその使用目的が異なる．

これは，今後の最大の研究課題の一つである．

3. 日本語コードの諸問題

3.1 字種の問題

日本語プログラミング環境を論じる際に，避けてとおれない問題に漢字の問題[8] がある．漢字の問題は，第一に字種が多いことである．漢字の字種の上限は大漢和辞典の 49,964 字，康熙(コウキ)辞典の 42,174 字であり，日本で日常使用されている字種は 1 万数千字程度である．実際 JIS[9] では，この中から使用頻度を考慮して6,353 文字の漢字のコードを定めた．すなわち，基本的な性質の漢字 2,965 文字を第一水準漢字集合と呼び，それ以外の漢字 3,388 文字を第二水準漢字集合としている．

ところが，実際の日本語情報処理では，量は少なくても人名をはじめとして，上記に含まれない文字コードが必ずといってもよい位に発生する．たとえば，最近は，どこの大学でも中国をはじめとするアジア各国からの留学生が数百人もいる．その場合，JIS コードに含まれない漢字が多く，ワープロで外字処理が必要となる．外字処理はユーザ個々により必要とする文字が異なるため，字種を制限した条件下で，意図に反して文書の互換性が崩れてしまうという問題が生ずる．もっと本質的な問題は，人文科学における計算機利用の一環としてのデータベースの構築などの研究では，外字処理が中心的な課題になってしまうことである．

この問題に関しては，たとえば JIS で第三水準，第四水準というように早急に設定した方が混乱が少ない．大漢和辞典の文字セットをすべて登録しても 16 ビットのコード系に収めることは可能だからである．それは，漢字を減らしていきたいという戦後の国語教育の方針との関係とは必ずしも矛盾するものではない．

3.2 情報処理上の問題

次にプログラミングにおける技術的な問題を考察してみると，現在の JIS の歴史的な成立過程からいろいろの問題が発生していることが分かる．

JIS コード（情報交換用符号）は，ASCII コード系（JIS 7 ビットコード系）の拡張という考え方が基本にある．また，設定時点の技術的制約条件と実現上の効率とをトレードオフをした結果も含まれている．たとえば，半角カナ文字と全角カナ文字という区別もその一つである．すなわち，記憶素子のコストが高い時代背景の下に設定されたコード系である．

現在流通している漢字コード系には，JIS 漢字コード (JIS X 0208：旧 JIS C 6226) とパソコン OS に採用されているシフト JIS コードがある．これらのコード系で共通していることは，1 バイトコードと 2 バイトコードとが混在していることである．これが情報処理における本質的な問題を含む原因となっている．まず具体的な事例を図-1 に示す．

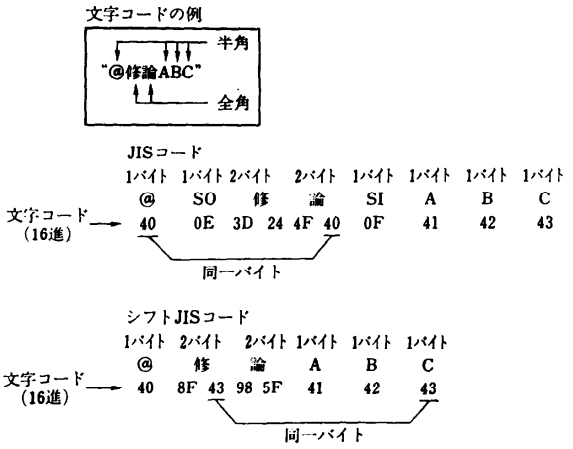

図-1 JIS とシフト JIS との内部表現の違い

JIS では，全角文字（2 バイトで表現）と半角文字（1 バイトで表現）が混在する文字列には，半角が主体であったから，全角コードへのシフトコード（シフトイン）が挿入され，再び半角コードへ戻るときにはシフトコード（シフトアウト）が挿入される．このようなコード系では，文字列を処理（たとえば文字列検索）する場合に，常に状態を意識し，先頭文字から順々に解析しなければならない．

一方，シフト JIS では，JIS X 0208 の漢字表の部分を，漢字コードの第一バイトが英数字と半角カナ文字と重ならないようにずらして（シフトして）配置した．その結果，少なくとも JIS のように，シフトコードを挿入しなくても英数字，半角カナ文字などと 2 バイト漢字コードとの区別はつけられる．しかし，2 バイト漢字コードの後半のバイトは半角コードと同一文字が現れるので，文字列を処理するには JIS コード系と同様に先頭から行わなければならない．シフト JIS の本質的な問題は，文字コードと属性との区別の概念がないことである．これでは，高度の情報処理には向かないことは自明である．

3.3 フル 2 バイト JIS コード系

現状の JIS 漢字コード（JIS X 0208）の情報処理上の問題を避けて，上位互換性を保つために，フル 2 バイト（16 ビット）コード系を提案する．この提案を検討してみよう．

文字コード系の設計では，大別して，文字コードの割当てを中心にしたコード系自体の問題と，文字属性の問題を考察する必要がある．

（1） 文字コード系について

（i） 文字（英字，数字，ひらがな，カタカナ，漢字，記号）のコードは JIS X 0208 のコード系を採用する．

前述のように，シフト JIS は，英字や数字の半角文字は 1 バイト，全角文字は 2 バイトに割り当てている．しかし，これは JIS に反するだけでなく同じ文字を違うコードに割り当てていることになる．これに対し，フル 2 バイトコード系では，コード体系の一貫性を重視し，一つの文字には一つのコードを割り当てるという原則に従う．

（ii） 制御コードも 1 バイト目に NULL (00) 16 を付加して 2 バイトにする

JIS X 0202 （旧 JIS C 6228）では，JIS X 0201 (旧 JIS C 6220) の 1 バイトコード系の中に JIS X 0208 の 2 バイト文字コードを組み込む方式を規定している．また，JIS X 201 において，NULL コードは情報内容に影響を与えず，どこにでも挿入できる規約になっている．したがって，ここで提案するフル 2 バイトコード系は JIS X 0201, JIS X 0202, JIS X 0208 の規定に従ったものである．これによって，2 バイトコードだけのコード系を構築する．

1 バイトコードと 2 バイトコードの混在はコード系の簡潔さを損なうだけでなく，プログラム上における文字，文字列の扱いを困難にする．たとえば，文字列へのポインタのインクリメントは，言語 C では

```
char *p;
**p;
```

と記述するが，1 バイト，2 バイト混在コードの場合には増分のバイト数は一意に決定することができなくなる．また，文字コードのバイト数は，文字コードの意味を調べなければ決定できなくなる．つまり，制御コードが 1 バイトで表現されていると言語 C で

```
static char *p="文字列\n";
```

と記述した場合には，改行符号（\n）は 1 バイトであるために，物理的には 7 バイト，論理的には 4 文字となり，実際には文字コードの意味を調べなければならないことになる．

（2） 文字属性について

日本語を扱う場合には半角文字などの文字属性をどのように扱うかが問題となる．特に，全角や半角などの文字の大きさに対する情報は，美観上必要となる．また，全角，半角，の違いは英語における小文字，大文字の違いとは基本的に異なる．日本語において全角や半角の違いは，あくまで外観上の違いだけである．これに対して英字における小文字と大文字の違いは，意味的な違いをも含んでいることに注意しなければならない．全角半角の違いは，英語で言えばむしろ 2 種類のフォントの違いに相当する．

これら文字属性の扱いはアプリケーションに大きく依存することに注意すべきである．したがって，拡張性に富んだ方法を考慮しなければならない．

これに対処するには，次の方法が考えられる．

（i） エスケープシーケンスを用いる

JIS で規格化されている文字属性（JIS X 0201, 0207）はエスケープコードによる状態遷移で行われる．たとえば，1B 5B 31 30 30 3B 35 30 20 42 というエスケープシーケンスは以降の文字を半角文字にする．この方式ではコードの 2 重定義が起こることはない．しかし，この方式の問題点は複雑なエスケー

プコードがテキスト中に混在してしまうことにある．アプリケーション・ソフトウェアは，この複雑なコードの存在を意識しなければならない．また，OS やコンパイラなどのシステム・ソフトウェアでは文字コード（すなわち，意味的な部分）のみを識別できれば十分であるのにもかかわらず，これらのコードを解析しなければならない．

（ⅱ） 文字コードに属性を表現するコードを付加する

文字コードと文字に対する属性のコードを1セットとして扱う方式である．この方式では，文字のもつ意味的な情報と書体やサイズなどの外観上の情報を一度にコード化できる．文字の属性を使用しない場合には文字コードのみを処理し，文字の属性が必要な場合には付加した文字の属性の情報を同時に処理すれば良い．しかし，文字を4バイトで表現すると文字の属性コードを必要としない場合にも常に4バイトごとにアクセスするために無駄が多くなる．また，文字属性の表現を一とおりの方式にコード化することは，かえって表現の拡張に制限を与える．つまり，文字の属性の表現を固定化（前の例では2バイト）すると，表現能力に限界が生じる．表現できる文字コードの数の拡張は2バイトで納まるが，文字の属性の表現では不足する可能性が高い．文字の属性は出力系のアプリケーションや出力デバイスの仕様に大きく依存している．文字コード系のアーキテクチャは，非常に長いライフサイクルをもつ必要がある．特に，アプリケーションや出力デバイスが多様化する傾向にあることを考慮すれば，拡張性の乏しいコード体系では満足なものとはいえない．

（ⅲ） 属性情報を別ファイルでもつ

文字コードだけのファイルと文字の属性情報や書式情報を別ファイルの形で表現する．このため，属性が必要なソフトウェアでは二つ以上のファイルを読み，その必要のないものは文字コードのファイルだけを読み込めばよい．この方式を用いれば，文字や文書の外観に関する情報から文字や文書のもつ意味的な情報を分離することが容易にできる．また，この方式では多様化するアプリケーションやハードウェアに対する拡張も容易である．たとえば，OS やコンパイラなどのシステムソフトウェアでは半角文字などの煩わしい文字属性を意識せずにすみ，エディタやフォーマットなどのアプリケーションソフトウェアでは豊富な文字属性を用いることが可能になる．

4. 日本語プログラミング環境

日本語プログラミング環境の主役は日本語である．既存の代表的なプログラミング環境の UNIX は，米国の計算機科学の学生や研究者にとっては自然な環境である．文字コードも ASCII に統一されているため，ネットワークでメイル交換しても，ネットワークファイルシステムを利用しても，プリントサーバを利用してもコード変換のことはまったく気にすることなく問題解決に専念できる．これはまさに快適な環境であるに違いない．

これに対し，同じ環境を日本人が利用すると，同上の米国人たちと同程度に英語で文章を書いたり話をしたりできる人の割合は低くなるため，可読性や保守性の悪いプログラムがつくられる割合が高くなる．そこで，「日本語 UNIX」という改造の努力[10]が行われる．しかし，しょせん，木に竹を接ぐのたとえのように，OS の一番深い約束ごとである“内部コード”を便宜的に拡張することで対応することになる．このような改造では，システム全体として，日本語化[11]はできない．日本語化を扱いたい箇所はすべて個別に対応することになってしまう．

すなわち，システム記述言語からエディタやファイルシステムすべてに一貫して日本語化を実現するには，文字コード体系から方式設計をし直さなければならない．

コード系の選択は，計算機システムの路線そのものであり，いったんそれで走り出すと，途中で変更できないことは，米国で ASCII コードが標準になっても，IBM はシステム/360 で標準化した EBCDIC コードを変更できないことでも明らかである．システム/360 以前には，IBM も事務処理用と科学用とで文字コードが異なっていた．システム/360 では，これを EBCDIC に統一した．

以下，本節では筆者らが開発している日本語プログラミング環境を指向した OS/omicron のアーキテクチャ[12)~14)]を例にとって，日本語プログラミング環境の開発経験と開発上の問題点を述べる．

4.1 システム記述言語とその言語処理系

OS/omicron は日本語情報処理への応用を考え，まず日本語プログラミング環境の構築を行うことにした．そのために，OS の内部コードとして，**3.3** で提案したフル2バイト JIS コード系を採用した．また，OS 自身を記述するシステム記述言語として言語Cを

ファイル名： demo.c 1989年 2月 18日 20時 47分 ページ 1

```
/*****************************************************************************/
/*              ファイル名:demo.c                                            */
/*              作成者    :大島                                              */
/*              作成日    :1989/02/17                                        */
/*(モジュール仕様) : 文書ファイル中の文字をクラスごとに分類するプログラム    */
/*****************************************************************************/

/***   ヘッダファイル          ***/
#include      (stdio.h)
#include      "ismacro.h"

/***   外部参照関数の宣言      ***/
FDCL_GLBL     INT             printf();
FDCL_GLBL     FILE            *fopenc();
FDCL_GLBL     INT             fclose();

/***   外部定義関数の宣言      ***/
FDCL_GLBL     AVOID           main();

/***   静的関数の宣言          ***/
FDCL_LCL      AVOID           文字分類();
FDCL_LCL      AVOID           文字数表示();

/***   外部静的データの定義    ***/
DLOCAL        FILE            *fp;
DLOCAL        INT             漢字文字数;
DLOCAL        INT             片仮名文字数;
DLOCAL        INT             平仮名文字数;
DLOCAL        INT             英字文字数;
DLOCAL        INT             数字文字数;
DLOCAL        INT             ギリシャ文字数;
DLOCAL        INT             その他;

/***   外部関数の定義          ***/
/*---------------------------------------------------------------------------*/
/*(関数名) : main                                                            */
/*(引数  ) INT     argc;    : コマンド行の引数の数                           */
/*(引数  ) CHAR    *argv[]; : 引数を含む文字列の配列を指すポインタ           */
/*(機能  ) : プログラムのエントリーポイント。                                */
/*---------------------------------------------------------------------------*/
ENTRY   AVOID   main(argc, argv)
INT     argc;
CHAR    *argv[];
{
        if (argc != 2)  {
                printf("使用方法 : [demo] [入力ファイル名]\n");
                exit(1);
        }
        if ((fp = fopenc(*++argv, 0)) == NIL)   {
                printf("入力ファイルがオープンできません！\n");
                exit(1);
        }
        文字分類();
        文字数表示();
        if (fclose(fp) ( 0)     {
                printf("入力ファイルがクローズできません！\n");
                exit(1);
        }
}
/***   静的関数の定義          ***/
/*---------------------------------------------------------------------------*/
/*(関数名) : 文字分類                                                        */
/*(機能  ) : 文字を分類して各クラスの文字数をカウントする。                  */
/*---------------------------------------------------------------------------*/
FLOCAL  AVOID   文字分類()
{
        CHAR    文字;

        while ((文字 = getc(fp)) != EOF)
                if (漢字判定(文字))     {
                        漢字文字数++;
                } else if (片仮名判定(文字))    {
                        片仮名文字数++;
                } else if (平仮名判定(文字))    {
                        平仮名文字数++;
                } else if (英字判定(文字))      {
                        英字文字数++;
                } else if (数字判定(文字))      {
                        数字文字数++;
                } else if (ギリシャ文字判定(文字))      {
                        ギリシャ文字数++;
                } else  {
                        その他++;
                }
        }
}
```

図-2 識別子に漢字を用いたプログラム例

採用した．OS を根底から日本語化するためには，まず，このシステム記述言語の処理系がフル2バイトコード系を扱えることが前提になる．しかし，そのような C の言語処理系は実在しない．そこでわれわれは，まず，そのような言語Cの処理系を開発[15]した．われわれはこの言語処理系を CAT (C compiler developed at Tokyo University of Agriculture and Technology) と呼ぶ．

CAT を実現すれば，われわれはCのプログラム中の識別子に日本語を自由に使用できる．識別子に日本語が使えるとなると，関数名，変数名に日本語が使えるし，日本語フォーマッタなどでは日本語で書かれている組版規則の用語を関数名やファイル名に使用できるようになる．この結果，プログラムの可読性は欧米と同じ水準に達せられる可能性がある．

識別子として許される文字は，英字('_' も含める)，平仮名，片仮名，第一水準漢字，第二水準漢字，ギリシャ文字であり，2文字目以降に数字を許す．

CAT の実現方式については紙面の関係で省略する．この日本語 CAT を使用した言語Cのプログラム例を図-2 に示す．

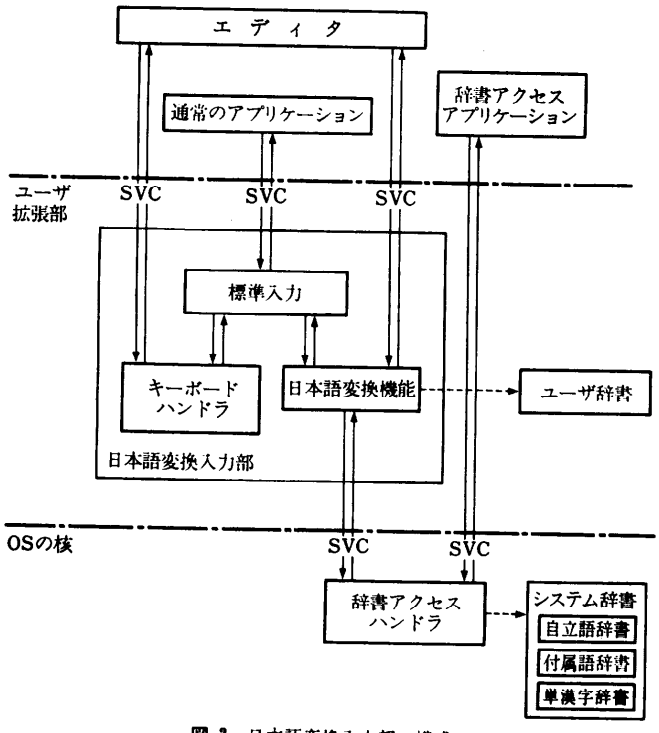

図-3 日本語変換入力部の構成

4.2 日本語の入力インタフェースとエディタ

システム記述言語とそのコンパイラを実現することによって，識別子やコメントに日本語が書けるようになったとしても，その入力手段がなければ絵に書いた餅に過ぎない．その意味で日本語の入力手段が重要である．

日本語プログラミング環境として，まず第一に用意する必要のある入力手段はキーボードからの仮名漢字変換機能とエディタである．特に，これらをオペレーティングシステムの標準的な入力手段としてもつことが重要である．また，これらの実現の際には，3.3 で述べたフル2バイト JIS コード系という内部コードと，属性ファイルとの組合せが満たされなければならない．このような構成の例を図-3 に示す．

この構成の利点は，OS の核の部分に，辞書のアクセスハンドラと辞書[16]（自立語辞書，付属語辞書，単漢字辞書，JIS 情報処理用語辞書）を用意していることである．したがって，辞書のアクセスハンドラをエディタからだけではなく，ユーザプログラムからでもスーパバイザコールによって利用できる．

また，ユーザ拡張部，すなわち，ユーザ固有 OS 拡張部に，システムの標準として仮名漢字変換系を用意する．この仮名漢字変換系には，フロントエンドプロセッサとしての機能のほかに，アプリケーションプログラムから変換する文字列を受け取り，その候補文字列を返すという機能をもたせる．このような機能をもたせることにより，ユーザが打ったキーをそのままエディタが受け取り，エディタから仮名漢字変換システムを呼び出すという形態の日本語エディタが実現できる．この方式では，エディタからみれば，日本語変換機能はエディタの1コマンドとして実現できる．

第二に，日本語プログラミング環境の一部として有効なのは，オフラインで日本語ワープロを活用する方式である．特に，最近のパーソナルワープロの価格性能比は抜群に高い．これらを用いて，設計仕様書やソースプログラムの初期入力をして，できあがったフロッピディスクを変換ツールを用いて，フル2バイト

JIS コード系に変換して利用する方式である.

三番目の入力手段としては,液晶表示付きタブレットなどを用いたオンライン手書き文字認識方式である.これは単にキーボードを使えない人たちの利用というよりも,CAIなどでの利用も含めて,手書きの文字に対応するものである.数学の CAI などを考えると,特殊記号をキーボードからの入力で行うのは,心理的負担が多く,好ましくないと考えるからである.

4.3 日本語の出力,DTP とプリントサーバ

日本語プログラミング環境として,入力,処理に続いて出力環境も重要である.

この部分は,ハードウェアの進歩に負うところが多い.初期のワープロは 24×24 ドットで漢字を表現していたが,レーザビームプリンタの出現で大きく改善されることになった.レーザビームプリンタは初期のものでも 10 ドット/mm の解像度であった.現在は業務用では 100 ドット/mm のものまである.

このようなレーザビームプリンタを中心とした,プリントサーバ(ネットワーク上で共有資源としてのプリンタ)や,ワークステーションを用いた DTP(卓上出版)の形態で,日本語のための環境を作る必要がある.

基本的な問題は次のとおりである.

(1) 印字品質の問題
(2) 禁則処理の問題
(3) 和欧混合組版の問題
(4) 図や表などのレイアウトの問題

このうち,(1)の問題は,ハードウェアの技術革新の問題とは別に,フォントの問題がある.日本語プログラミング環境としては,活字の号数に対応する文字の大きさをすべて,ドットフォントでもったのでは能率が悪い.ベクトルフォントや輪郭線フォントからの生成の研究が必要である.

次に(2)の禁則処理に関しては,現行のワープロではぶら下り組による禁則処理しか実現されていない.行頭禁則処理や行末禁則処理においては一般の書籍レベルを実現する必要がある.これは,プリントサーバや DTP による組版規則の実現とも関連する.

和欧混合組版[17]こそ,明治以来,日本の出版業界が取り組んできた課題であった.日本の文章の中には,仮名と漢字ばかりでなく,英単語や,英語のフレーズなどが含まれる.これを自然な形に組むために,和欧混合組版用の英字活字の開発や,英字の部分のプロポーショナルピッチの採用,文字間隔と行間隔とのバランスを活字の大きさによって許容範囲の値を表形式で提示するなど,非常に細かい組版規則をもっている.これらは,一意的に解が求められる規則ではなく,人間の経験に支えられている.したがって,"知的和欧混合組版"の実現が課題となる.

次に(4)の図や表のレイアウトの問題は,文書の構造論からのアプローチや,割付けのアルゴリズム,図中のテキスト部分の処理,参考文献の文献データベースからの作成と編集など,情報工学としては寄与できることが多い部分である.

最後に,「美しいプログラムリスト,文書の出力は,プログラマの創作意欲を高める」[18]ことを締めくくりとしておく.

5. 知的日本語プログラミング環境への指向

日本語プログラミング環境の目標は,自然言語としての日本語でのマンマシンインタラクションの実現である.

現在のところ,プログラミング言語をとってみても,日本語の構文規則にのっとり,多様な表現を許すようなもので,多くのユーザから支持を集めるような言語はできていない.ある意味で,永遠の課題であり,一歩一歩前進させていく研究であり,機械翻訳と似た状況といえる.

計算機を自然言語でアクセスする試み[19]の一つとして,システム・メッセージやコマンドにその国の国語を用いる研究がある.たとえば,システム・メッセージに対して内部表現として一つ用意しておき,対象とする国語に対する変換系を用意するというものである.

いずれにせよ,自然言語,あるいは自然言語に制約をつけた自然言語ふう人工言語を扱うためには,その自然言語(日本語の場合には日本語)自身の息の長い研究が必要である.すなわち,自然言語のコンピュータによる理解の研究や,常識という知識ベースの構築などの研究の積み重ねが必要である.日本語プログラミング環境はこのような研究のための環境でなくてはならない.

より,現実的な研究目標として,知的日本語プログラミング環境を実現するためには,次の研究が必要である.

(1) 日本語による文書化の作業を助ける研究

この研究の一環として,作文支援系の開発や,コンピュータによる辞書システムなども含まれる.

（2） 日本語による“使い方”についての質問応答機能の研究

この研究の一環として，知的 help 機能とか知的 CAI なども含まれる．

（3） 日本語によるプログラミングの研究

仕様書からのプログラム合成などを実現するための基礎的研究などが含まれる．

（4） 出版物などの光学的読取りなどの入力技術

（5） 知識ベースの構築

あえて，(4)を問題とする理由は，人工知能の研究や，データベースの研究は，ある量（相当な量）のデータの蓄積がないと，有効な結果を引き出せないからである．コンピュータによる処理の中では，依然として，入力は人間によることが多い．(5)を実現するには(4)を利用するという戦略もあるが，ただちに実現できない．したがって現実は，ブルートフォースで知識ベースを構築することになるだろう．

6. おわりに

日本語プログラミング環境の実現は，オペレーティングシステムの内部コードから，それを用いる言語処理系，ファイルシステム，データベースに至るまで，一貫したアーキテクチャが必要になる．

筆者らは，OS/omicron（第2版）という日本語対応のオペレーティングシステムの開発を完了した．今後，これを利用して上記の問題に取り組む予定である．

参考文献

1) Wilkes, M. V. et al.: The Preparation of Programs for an Electronic Digital Computer, 2nd ed., Addison-Wesley Publishing Company, Inc., U. S. A., p. 238 (1957).
2) IBM: Programmer's Primer for FORTRAN Automatic Coding System for IBM 704 (1957).
3) CODASYL: COBOL-A Report to the Conference on Data System Languages, including Initial Specifications for a Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers, Government Printing Office (1960. 4).
4) 高橋延匡：日本語情報処理システムへの期待，情報処理，Vol. 20, No. 1, pp. 44-49 (1979).
5) 高橋延匡：日本文入力の現状と展望，情報処理，Vol. 23, No. 6, pp. 518-528 (1982).
6) 高橋延匡：5年後の日本語プログラミング環境，情処シンポジウム論文集，Vol. 88, No. 2, pp. 81-92, 情報処理学会 (1988. 4).
7) 並木美太郎，関口 治，里山元章，中川正樹，高橋延匡：日本語ワードプロセッサのソフトウェア生産への応用，情報処理学会ソフトウェア工学研究会資料 47-1, pp. 1-8 (1986. 5).
8) 高橋延匡編，石綿敏雄，西村恕彦，田中穂積，菊池光昭，藤崎哲之助：日本語情報処理，近代科学社 (1986).
9) JIS ハンドブック，情報処理-1988, 49, 50, 51, 52, 日本規格協会 (1988).
10) 中原 康，佐藤忠幸，秋山和浩，山田 操：UNIX 環境下での日本語処理，コンピュータ・システムシンポジウム論文集，pp. 193-202 (1985. 12).
11) 高橋延匡，大場 充，勝村哲也，野口健一郎，三好 彰，和田英一：OS と日本語サポート（パネル討論），情処シンポジウム論文集，Vol. 87, No. 2, pp. 159-168, 情報処理学会 (1987. 11).
12) 田中泰夫，中川正樹，高橋延匡：日本語プログラミング環境，究極のプログラミング環境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，pp. 69-79 (1987. 7).
13) 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡：OS/omicron における日本語プログラミング環境，情報処理学会論文誌，Vol. 30, No. 1, pp. 2-11 (1989).
14) 高橋延匡：研究プロジェクト総説：OS/omicron の開発，情報処理学会オペレーティングシステム研究会資料 39-5 (1988).
15) 並木美太郎，屋代 寛，田中泰夫，篠田佳博，藤森英明，中川正樹，高橋延匡：OS/omicron 用システム記述言語C処理 Cat のソフトウェア工学的見地からの方式設計，電子情報通信学会論文誌 D, Vol. J 71-D, No. 4, pp. 652-660 (1988).
16) 吉田 将：「日本語単語機械辞書」について，文部省科学研究費特定研究「情報化社会における言語の標準化」総括班 (1984. 2).
17) 写研・写研ルール委員会：組み NOW，(株)写研 (1975).
18) 里山元章：日本語文書出力システム「浄書」の基本設計とシステムの実現，ヒューマンフレンドリーなシステム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，pp. 181-193 (1986. 7).
19) 大平 剛，大場 充：オペレーティング・システムの他国語サポート，オペレーティング・システム研究会報告 No. 33, 情処研報，Vol. 86, No. 84, pp. 1-7 (1986. 12).

（平成元年2月23日受付）