プログラミング 7－3
（1996．5．31）

ポータブルな言語処理系の日本語化方式

牛嶋 哲
東京工業大学 理工学研究科 情報科学専攻

広く普及している既存の上級言語に変換する言語処理系を日本語化する際の問題点および実装例について述べる．このような処理系の日本語化に当たっては中間言語の処理系における日本語処理機能を仮定したり，特定の実行環境に依存してはならない．文字の取り扱いに関する3階層（符号層・文字集合層・文字概念層）を認識した上で，（1）複数の符号系への対応，（2）日本語文字用のデータ型および（3）文字の同一性について考慮する必要がある．実装例としてCLUをCに変換する処理系CLU2Cを日本語化した．各種の言語処理系について同様の手法が適用できると考える．

A Japanization Method for Portable Implementations of Programming Languages

Tetsu Ushijima
Department of Information Science, Tokyo Institute of Technology

General considerations and a sample implementation for Japanization of programming language processors that translate to an existing widespread high-level language are described. In Japanization of such language processors, one cannot depends on a Japanese processing facility in the intermediate language and a particular execution environment. On the understanding that there are three layers (encoding layer, character set layer, and character concept layer) in character handling, (1) multiple coding systems, (2) data types for Japanese characters, and (3) identity of characters must be considered. As a sample implementation, Japanization of CLU2C, which translates CLU to C, is described. Similar technique is also applicable to other language processors.

1 はじめに

ポータブルな言語処理系を作るための一つの方法としては，抽象化レベルに関して原言語との隔たりが小さく，多くの機種に処理系が存在するような既存の上級言語を中間言語として使用することが考えられる．最近では Eiffel, Sather, Pascal, Fortran, Scheme 等について，中間言語として C を用いる処理系の実例がある [1, 2].

一方日本語を母語とする者としては，たとえば図 1 のようにではなく，むしろ図 2 のようにプログラムを書きたい．つまり，

- 原プログラムの構成要素（識別子・リテラル・注釈等）における日本語文字の使用を許す．
- プログラムが扱うデータとして日本語文字が自然に扱えるようにする．

といった言語処理系の日本語化が必要である．上のやり方で作成するポータブルな処理系においてこれらの要求事項を満たそうとする場合には，

- 中間言語の処理系は特に日本語化されていない．
- 特定の実行環境（たとえばファイルはすべてシフトジスによって符号化されている，など）に依存してはならない．

という前提を置く必要がある．

これに対して，日本語を含む名前は一定の規則に従って英数字列に変換し，中間言語の処理系が日本語に対応しているかどうかに依存しないようにする，という方法が考えられる．また，日本語をデータとして扱うためには，日本語テキスト処理の基本的な部分を実行時システムに押し込めることが考えられる．ただしそれには言語処理系の日本語化に付随する問題点，言語仕様および実装方法について注意深く考慮する必要がある．

以上のことについて論じ，実装例として CLU 言語 [3] の処理系 CLU2C[4] の日本語化について述べる．各種の言語処理系および各国語について同じことが可能であると考える．

```
% Converts punctuation symbols.
% Assumes Japanese EUC.

conv_punct = proc(input, output: stream)
  while true do
    c1: char := stream$getc(input)
    stream$putc(output, c1)
    if c1 = '\241' then
      c2: char := stream$getc(input)
      if c2 = '\242' then          % touten
        c2 := '\244'               % comma
      elseif c2 = '\243' then      % kuten
        c2 := '\245'               % period
      end
      stream$putc(output, c2)
    end
  end
    except when end_of_file: end
end conv_punct
```

図 1: CLU によるプログラム例

2 日本語化に当たって考慮すべき点

2.1 文字の取り扱いに関する階層

言語処理系の日本語化とは，一般的には，プログラムが扱うデータとしての文字および原プログラムを構成する文字の集合として，単一の文字集合ではなく複数の文字集合を組み合わせたものを言語処理系が取り扱うことであると考えられる．この場合，文字の取り扱いに関して以下に示す 3 段階の階層があるということを認識する必要がある（**表 1** および図 **3** 参照）．

- **符号層**: 文字の列をバイト列（符号）として取り扱う．文字の列とバイト列の対応（符号系）は 1 種類とは限らず，日本語に関しては少なくとも ISO-2022-JP[5]，日本語 EUC およびシフトジスという 3 種類の符号系が用いられている．
- **文字集合層**: 各文字を，その文字が属する文字集合およびその文字集合内の識別情報の組として取り扱う．この層における文字の取り扱いは符号系に依存しない．
- **文字概念層**: 各文字が持つ概念的意味に基づいて文字を取り扱う．たとえば ASCII の「$」お

```
% 句読点をピリオド・コンマに変換する.

句読点を変換 = proc(入力, 出力: stream)
  while true do
    文字: char := stream$getc(入力)
    if 文字 = '、' then
      文字 := ', '
    elseif 文字 = '。' then
      文字 := '. '
    end
    stream$putc(出力, 文字)
  end
    except when end_of_file: end
end 句読点を変換
```

図 2: 日本語対応版 CLU によるプログラム例

表 1: 文字表現の階層

符号層	文字集合層	文字概念層
24	「$」 (ASCII)	「$」
1b 24 42		
24 48	「と」 (JIS X 0208)	「と」
21 70	「$」 (JIS X 0208)	「$」
24 4f	「は」 (JIS X 0208)	「は」
46 31	「同」 (JIS X 0208)	「同」
24 38	「じ」 (JIS X 0208)	「じ」
1b 28 42		
3f	「?」 (ASCII)	「?」

よび JIS X 0208 の「$」は文字集合層では別個の文字であるが，文字概念層では同一の「ドル記号」として取り扱うことが考えられる．ただしこの層の正確かつ一般的な定義を与えることは難しい．

2.2 複数の符号系への対応

ポータブルな言語処理系を前提とする（つまりその処理系をいろいろな環境に持ってゆきたい）のであれば，どの符号系を使うのかを限定してしまうのは得策ではない．一方，プログラマの立場からすると符号層と文字集合層の間の変換処理を一々記述するのは煩わしい．この処理は処理系が提供する入出力モジュールで行なうべきである．こうすることによってプログラマは符号層の存在

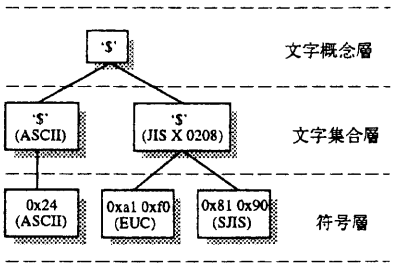

図 3: 文字の同一性

を通常は意識せずにプログラムを記述することができる．新たに別の符号系を使用したい場合には入出力モジュールを変更すればよい．

2.3 日本語文字を扱うためのデータ型

日本語文字および日本語文字列を扱うための基本的なデータ型を提供するためには，

- 既存の文字型・文字列型を拡張して日本語文字が扱えるようにする
- 既存の型とは別の型として，日本語文字用の文字型・文字列型を新たに導入する

という 2 種類の方法が考えられる．

日本語文字を普通の文字として扱いたいのであれば既存の型を拡張する方法を採るべきである．新しい型を導入すると，文字の扱いについて不必要な制約が課せられることになる．つまり従来の 1 バイト文字だけを扱うのかそれとも日本語文字を扱う可能性があるのかを決めた上でプログラムを記述しなければならない．

ただし以下のように既存の型を拡張することが難しい場合もありうる．

- 互換性を保つことが難しい
- 効率の低下が許容できないほど大きい．

このような場合には新しい型の導入を検討した方がよいかもしれない．

2.4 複数の文字集合における文字の同一性

前述したように文字概念層を正確かつ一般的に定義することは難しいので，文字をデータとして扱う立場では文字を文字概念層で取り扱うことにしてしまうのは賢明ではない．

一方，原プログラムを構成する文字については

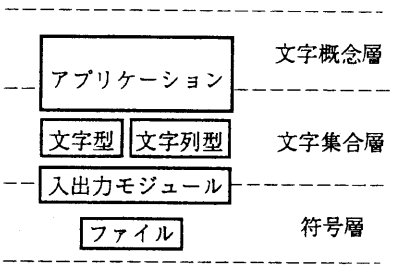

図 4: 日本語に対応した処理系における文字型，文字列型，および入出力モジュールの位置付け

多少事情が異なる．たとえば日本語入力システムおよび表示装置によってはASCII文字およびJIS X 0208の文字の間で混同が生じる場合がある(典型的な例は間隔文字であろう)．したがって少なくともASCIIに含まれる文字はそれと等価なJIS X 0208の文字と同一の文字概念を持つものとし，原プログラムを構成する文字は文字概念層で扱うことにする方がよい．ただし文字リテラルおよび文字列リテラルについては文字をデータとして扱う立場に準ずるべきである．

2.5 望ましい位置付け

以上の考慮点をまとめると，日本語に対応した言語処理系では文字型，文字列型，および入出力モジュールは図4に示す位置付けにするのがよいと思われる．つまり入出力モジュールがファイル入出力において符号層と文字集合層の間の変換を担当し，文字型および文字列型は文字集合層で文字を扱うのである．アプリケーションプログラム（たとえばここで日本語化しようとしている処理系）は基本的には文字集合層で文字を扱い，必要に応じて文字概念層との間の変換を行なう．

3 CLU言語処理系CLU2Cの日本語化

実装例としてCLU言語処理系CLU2C[4]の日本語化について述べる．CLU2CはCLUをCに変換して処理する処理系である．この言語および処理系を対象としたのは，この処理系が筆者らのグループが開発したものであり学部教育にも有効に使われている[6]という理由によるが，他のポータブルな言語処理系についても同様の方式が適用できると考える．

表 2: 現在使用できる符号系の例

文字列表現	符号系
"iso-2022-jp"	ISO-2022-JP
"euc-japan"	日本語EUC
"sjis"	シフトジス
"noconv"	(符号系を考慮しない)
"autoconv"	(符号系を自動判別)

3.1 CLU言語の仕様変更

CLU言語の仕様はASCIIを前提としたものになっているので，ある程度の仕様変更はやむを得ない．以下に言語仕様の変更点を挙げる．

- 原ファイルを構成する文字の集合としてJIS X 0208を追加し，原ファイルを構成する文字は文字（列）リテラル以外を除き，文字概念層で解釈されるものとした．
- 識別子，リテラルおよび注釈に日本語文字を含めても差し支えないものとした．
- 組み込みの文字型・文字列型を拡張し，日本語文字を含むことができるものとした．
- 文字（列）リテラルにおいて，\x*hhhh*という形式の拡張表記を用いて日本語文字を番号で指定できるようにした（16進4桁で指定する）．
- 入出力モジュールを拡張し，読み書きするファイルの符号系を属性として持つものとした．参考として現在符号系の指定として使える文字列の例を表2に示す．

3.2 字句解析部の変更

字句解析部はCLUで記述されているので，原ファイルを文字集合層で取り扱うことができる．必要な変更は以下の2点であった．

- 原ファイル中の文字を文字概念層で扱うために，日本語文字用の表を字句解析部内に用意した．
- 日本語文字が許される要素を正しく処理するために，一部の字句切り出しルーチンを変更した．

表 3: コード生成部が構成した識別子の例

CLU の構成要素		対応する C の識別子
名前	種類	
po	変数	LVpo
		整 列
整列	手続き	AF_X40304e73

3.3 コード生成部の変更

3.3.1 識別子の処理

CLU の識別子をもとに C の識別子を構成する場合，CLU の識別子に含まれる日本語文字を英数字列に変換する必要がある．そこで，たとえば CLU の手続き「整列」に対応する C の関数の名前が AF_X40304e73 となるようにした（表 3）．先頭の AF は，この名前が手続きに対応する関数の名前であることをあらわしている．これは日本語文字を含む識別子の処理とは特に関係はない．次の_X は，その後に文字の番号を 4 桁の 16 進数で表記したものが続くことを意味する．

この方法の 1 つの問題点は構成される C の識別子が長くなることである．しかし多くの C 処理系ではかなり長い識別子が使用できるようになっている．したがって実用上困ることは少ない．

なお日本語文字を含む識別子を英数字列に変換するという方法自体は，C 言語用の前処理プログラムを作った例がある [7].

3.3.2 リテラルの変換

文字リテラルについては，日本語文字は整数による表現を用いるようにした．文字列リテラルについては，リテラルの値に初期設定される C の静的変数の宣言を生成し，その変数を用いてリテラルの値を参照するようにした．

3.4 組み込み型および標準ライブラリの変更

3.4.1 文字型および文字列型の変更

従来の CLU2C では，文字型および文字列型の内部表現として C における表現をそのまま用いていた．JIS X 0208 が規定する符号の単位は 2 バイトなので，このままでは日本語に対応することができない．そこで，文字は C における int であらわすことにし，ただし下位 16 ビットのみが有効であるものとした．また文字列については 2 つのメンバで構成される構造体を定義し，第 1 メンバ (int 型) には文字数を，第 2 メンバ (short int 型の配列) には文字の並びを格納することにした (図 5 参照)．また，文字の番号づけは以下のように行なった．

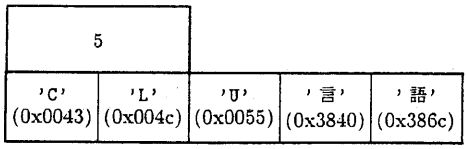

図 5: 文字列 "CLU 言語 " の内部表現

- ASCII 文字については，文字コードをそのまま番号とする．
- 日本語文字については，JIS X 0208 で規定される 7 単位符号系における 2 バイト符号を整数に変換したものを番号とする．

操作の実装は，文字型に関してはほとんど変更する必要はなかったが，文字列に関してはほぼ全面的に書き直した．これは従来利用していた C の文字列操作ライブラリ（strlen，strcmp など）が利用できなくなったためである．

3.4.2 入出力モジュールの変更

従来の入出力モジュールは C の標準入出力ライブラリを用いて実装されていた．そこで，まず C の標準入出力ライブラリを用いて C の標準入出力ライブラリと同様のインタフェースを持つ日本語入出力ライブラリを作成した [8]．次に C の標準入出力ライブラリを用いている部分を日本語入出力ライブラリを用いるように変更した．

4 日本語対応版 CLU2C の性能評価

処理系の日本語化に伴ってその処理系を用いて実行されるプログラムの実行効率がどのように変化するのかを，CLU2C について調べた．

表 4: 測定結果

処理内容	実行時間 (秒)		比
	日本語非対応版	日本語対応版	
入出力	1.08	2.80	2.59
記憶域	9.30	10.64	1.14
文字列	3.48	4.03	1.15
その他	14.84	15.03	1.01
計	28.7	32.5	1.13

4.1 測定方法

ある程度大きなプログラムにある程度の量を持つ入力データを与えてその実行時間を計測する，ということをした．具体的には，CLU2C のセルフコンパイルを行なった．

測定に当たって，日本語対応版および日本語非対応版（日本語対応版をもとに，文字型，文字列型および入出力モジュールを従来の実装で置き換えたもの．ただし文字列型については内部表現は図 5 のまま，文字の幅を 1 バイトとした）という 2 つの版の CLU2C を用意した．

まず，これら 2 種類の CLU2C を用いて日本語対応版 CLU2C の原プログラムから 2 種類のコンパイラを作成する．そして，これら 2 種類のコンパイラで日本語対応版 CLU コンパイラの原プログラム（約 2 万行）を C に変換するときに要する時間を測定した．

測定に用いた環境は次のとおりであった．計算機は Sony NWS-5000 (NEWS-OS 4.2.1R) を使用した．CLU プログラムを C プログラムに変換する際コンパイラの最適化オプションを指定した．C 言語処理系は OS に付属のものを用い，最適化レベルを 2 とした．

4.2 測定結果

測定結果を表 4 に示す．ただし処理内容別の実行時間は，各関数を入出力処理，記憶域管理，文字列処理およびその他に分類し，各処理内容の実行サイクル数を比例配分して求めたものである．

実行時間の増加がもっとも大きいのは入出力に関する処理である．入力・出力ともに増加分の大半は符号変換を行なう部分で消費されている．次いで大きいのは記憶域管理および文字列処理に費やされた時間であった．これは文字列オブジェクトが一様に大きくなったことによる増加である．

5 結び

日本語プログラミングに対する要求事項，その実現に当たっての問題点，および CLU 言語処理系 CLU2C における日本語化の実装例について述べた．2 節の議論は CLU 以外の言語にも適用できるはずである．また，3.4.2 で述べた日本語入出力ライブラリは特に C を中間言語とする処理系の日本語化に役立たせることができる．

参考文献

[1] Meyer, B.: Eiffel: Programming for Reusability and Extendibility, *ACM SIGPLAN Notices*, Vol. 22, No. 2, pp. 85–94 (1987).

[2] Bartlett, J. F.: SCHEME->C a Portable Scheme-to-C Compiler, WRL Research Report 89/1, DEC Western Research Laboratory (1989).

[3] Liskov, B. and Guttag, J.: *Abstraction and Specification in Program Development*, MIT Press (1986).

[4] 江原善: 開発効率と移植性を重視した CLU 言語処理系の作成, 修士論文, 東京工業大学 (1992).

[5] Murai, J., Crispin, M. and van der Poel, E.: Japanese Character Encoding for Internet Messages, RFC 1468 (1993).

[6] 木村泉, 大野浩之: ソフトウェア専門家を志す人のための基礎訓練科目, 情報処理学会論文誌, Vol. 34, No. 12, pp. 2449–2457 (1993).

[7] 平林雅英: C 言語の日本語化とその効果, 情報処理学会研究報告, 88-PL-16 (1988).

[8] 牛嶋哲: 標準 C ライブラリと同様のインタフェースを持つ日本語入出力ライブラリの開発および評価, 第 52 回情報処理学会全国大会講演論文集 (3), pp. 261–262 (1996).