プログラミング言語 16 － 3
（１９８８．５．１３）

日本語 APL

宇土正浩，佐貫俊幸

日本アイ・ビー・エム株式会社
東京基礎研究所

近年日本語のデータに関する処理を可能にするため，プログラミング言語に日本語処理機能を導入することが重視されている．筆者らは，この日本語処理機能を APL 言語に導入し，パーソナルコンピュータの環境で稼働する APL 言語システムを開発した．このシステムの特徴は従来ホストシステムで利用可能であった APL をパーソナルコンピュータの上で実現した点，および従来の VS-APL との互換性を持たせつつ漢字データを英数字と同等に扱えるようにした点である．さらにパーソナルコンピュータの資源を APL から利用するための補助プロセッサ群もこの特徴の一つとして挙げられる．本論文では APL 言語に漢字データ処理機能を導入する際の言語処理系の設計とその実現方法について述べる．更に現在研究を行っているワークステーション上の APL2 において，多国語処理を踏まえた日本語処理の方法についても言及する．

Japanese APL - Its Design and Implementation

Masahiro Udo and Toshiyuki Sanuki

Tokyo Research Laboratory, IBM Research
5-19, Sanban-cho, Chiyoda-ku, Tokyo 102, Japan

This paper exemplifies the national language support of a programming language by describing the design and the implementation of APL with a language facility for manipulating Japanese characters. By national language support, in this paper, we mean that users are enabled to use their own national language in the data, as legal character strings of the programming language. We describe the implementation of APL that provides Japanese language support in the above sense on IBM Personal System/55. We also discuss the design principle adopted to include a new two-byte character set in APL data objects coexisting with a conventional one-byte APL character set. Moreover, we introduce an exploration of Japanese character enabling based on multinational character sets, which is introduced in a prototype of APL2 on workstation.

はじめに

対話式汎用プログラミング言語の APL は，その基本データ・オブジェクトとして配列データを用いている．このため数値データおよび文字データをスカラーあるいは N 次元の行列として自由に扱える特徴を持つ．この機能は他のプログラミング言語と大きく異なる点であるが，それゆえに APL において扱う文字データに新たに日本語の文字を加えるにあたって，多くの配慮を必要とする．現在，COBOL や FORTRAN 等の標準プログラミング言語では，日本語処理機能として漢字 (2 バイト) データを扱える処理系が存在している．しかし上記のごとく APL は配列指向の言語のため単純な文字ストリングの操作だけでは，APL の持つ多くの原始関数には対応できない．また APL は作用素や原始関数を表すために特殊な記号を多く用いるが，既存の文字とどのように折り合わせるかも大きな問題となっている．さらに APL はインタープリタ形式で関数を実行するため，日本語データを処理するライブラリを導入することができない．従来からある APL 言語との互換性を保ちながら，新たに日本語のデータを APL 言語でどのように取り扱ったらよいのか検討し，筆者らの行った方法について具体的に述べる．また，現在研究を行っているワークステーション上の APL2 における多国語処理を踏まえた日本語処理の方法についても触れてみる．

言語仕様の設計

設計指針

従来の英数字 (1バイト文字) も漢字あるいは仮名 (2 バイト文字，以下"日本字"と記す) もデータとしては 1 つの文字として意味を持ち，決して日本字を 2 要素からなるデータとして扱うのは得策ではない．これは日本字に限ったことではなく，中国語や韓国語などでも同様なことがいえる．また文字のエンコーディング方法を絶えずユーザに意識させながらプログラミングを行わせるのはユーザの負担になる．筆者らはこの観点から APL 言語に日本字データを導入するにあたり，次の指針を設けた．

1. 1 つの日本字データはたとえ 2 バイトでエンコードされていても，あくまで 1 個の文字スカラーとして意味を持つのであり，2 要素からなる文字ベクトルとして扱わない．これを例えばベクトルの要素数を得る原始関数 ρ で示すと，

```
      ρ'漢字'
2
```

であり，

```
      ρ'漢字'
4
```

とはしない．

2. 文字配列に対して定義できる原始関数は全て日本字のはいったデータにも対応出来るようにする．さらに，日本字データは従来の英数字データと同等に扱えるようにする．例えば，

```
      'A' = 'APL'
1 0 0
```

と同様に，

```
      'A' = '日本語APL'
0 0 0 1 0 0
```

あるいは，

```
      '本' = '日本語APL'
0 1 0 0 0 0
```

となる．

3. 日本字はリテラル定数の要素として扱えるようにする．変数名やプログラム名などの識別子にも日本字を使えるようにすることは，処理系の負担を大きくさせるばかりでなく，プログラムの移植性，可搬性を落とす可能性が高くなるため適当でないと判断した．また，現在の漢字入力方法を考えると英数字文字の入力に比べはるかに手間がかかるため，このレベルまで日本字をサポートすることは余り効果がないと判断した．

4. APL言語の標準化および互換性を維持するため，日本字データ処理用の新な原始関数あるいはシステム関数を導入することは避ける．

以上 1. から 4. の指針に従い，APL 内部における日本字データの具体的な処理方式について考察した．

日本字処理における APL 固有の問題

従来の APL の処理系では，APL オブジェクトのタイプは次の 4 つから成っている．

論理値	(1 ビット/ 要素)
整数	(2 バイト/ 要素)
浮動小数点	(8 バイト/ 要素)
文字	(1 バイト/ 要素)

2バイトで表現される日本字を上記の文字型データタイプに適用すると，以下の問題が生じる．

- 日本字に適用できない原始関数が存在する．

  例えば，

```
'日本語APL' = '語'
```

  はLENGTH ERRORとなってしまう．これは漢字 1 文字を 2 バイトの要素からなる文字ベクトルとして扱っているために，右辺と左辺のベクトルの長さが一致せずエラーを起こした．これを解決するためには，スカラーとベクトルの差を正しく維持することが必要である．つまり“語”と言う漢字データをスカラーとして扱えるようにすれば，この問題は解決する．

- 日本字が他の字に化ける．

  日本字は 2 バイトのコード体系からなっているが，その第一バイトと第二バイトの組合せは一般的に非可換である．つまり，“語”はシフト JIS コードでは x“8C-EA” であるが，第一バイトと第二バイトを反転すると x“EA-8C” となり，このコードは“鑑”を表す． APL ではデータ構造の配置を変える原始関数が多く存在するが，日本字データもこれらの関数に適応できなければならない．例えば要素の回転を行う原始関数⌽ (Rotation) は，

```
⌽'日本語APL'
⌽|93|FA|96|7B|8C|EA|41|50|4C|
  |4C|50|41|EA|8C|7B|96|FA|93|
    L  P  A  鑑  {  愈
```

となり，日本字の2バイトの対を壊してしまい，正しい日本字の表現を乱してしまう．またシフト JIS コードでは， 2 バイトコード文字の第一バイトは x“81” から x“9F” および x“E0” から x“FC” の範囲に入っていなければならないが，次の例では第一バイト目が落ちたために漢字が正しく表現されていない．

```
5↑'日本語APL'
5↑|93|FA|96|7B|8C|EA|41|50|4C|
    |EA|41|50|4C|
      鑑  P  L
```

これらの問題を解決するには，日本字がデータの中に入った場合， 2 バイト単位のデータ列として扱う必要がある．上記の例に対してこの考えを導入すると次のようになる．

```
⌽'日本語APL'
⌽|93FA|967B|8CEA|4100|5000|4C00|
  |4C00|5000|4100|8CEA|967B|93FA|
    L    P    A    語    本    日

5↓'日本語APL'
5↓|93FA|967B|8CEA|4100|5000|4C00|
  |4C00|
    L
```

処理系の拡張

前節で述べた日本字の処理方式を，従来のAPLの処理系に直接適用することは，1バイト単位の APL文字データ・タイプでは困難であり，新に APL のデータ・タイプを拡張する必要がある．

そこで筆者らは APL の文字データのタイプを，1バイト単位表現と 2 バイト単位表現の 2 種類持たせるようにした．前者は従来の英数字 (1 バイト) のみのデータに適用し，後者は日本字あるいは日本字混じりのデータに適用することを決めた．従って後者の場合には 1 バイトの英数字も 2 バイトの表現を採る．すなわち下記のようなデータ・フォーマットを用いている．

- 1 バイト文字のみのデータ：

```
      'ABCdef123'
←→|41|42|43|64|65|66|31|32|33|
```

- 2 バイト文字のみのデータ：

```
      'かな漢字'
←→|82A9|82C8|8ABF|8E9A|
```

- 1 バイト・ 2 バイト混在データ:

```
      '漢字ABC'
←→|8ABF|8E9A|4100|4200|4300|
```

このようなデータ・タイプを導入すると，次のような混在データの連結 (Catenate) においては，タイプの変換処理を行わなければならない．

```
      Dat←'ABC'
      Dat←'漢字',Dat
      Dat
漢字ABC
```

しかしこの変換は APL 内部では日本字データ固有の処理方式ではない．数値データの場合においても，整数と実数の間での変換と同等である．すなわち

```
      Var←1  2  3
      Var←0.5 0.7,Var
      Var
0.5 0.7 1 2 3
```

の連結においても，1 行目では 1 要素 2 バイトで表現されている整数のベクトル Var は，2 行目の，によって 1 要素 8 バイトで表現される浮動小数点のデータ・タイプに変換される．

日本語 *APL* の実現

筆者らは，IBM-PC 上の APL(PC/APL)を元に，前章で述べた方式により，APL に日本字データの処理機能を導入し，IBM マルチステーション 5550 の日本語 DOS の上で “日本語 APL ”を実現した．日本語APLの構成は以下の 3 つの部分から構成される．

- 入力されたステートメントを解析し，評価・実行するインタープリタ．
- マシンやオペレーティングシステムの DOS に依存しながらユーザとの対話処理を行ない，またワークスペースの管理を行うエクゼキュータ．
- 周辺機器や DOS ファイル等のシステムに依存する資源を APL から利用可能にするための補助プロセッサ．

インタープリタ

従来の英数字と同様に日本字データを，APL の原始関数や作用素に対応させるためには，インタープリタ自体に先に述べたデータ・タイプを扱えることが必要である．

筆者らは PC-APLのインタープリタを元に，2 バイトで 1 要素を表現する日本字のデータ・タイプを文字データの拡張として導入した．すなわち日本語 APL のインタープリタは次のデータ・タイプを扱えるようにした．

論理値データ	(1 ビット/ 要素)
整数	(2 バイト/ 要素)
浮動小数点	(8 バイト/ 要素)
バイト単位文字	(1 バイト/ 要素)
ワード単位文字	(2 バイト/ 要素)

これにより全ての原始関数とシステム関数は日本字の含まれたデータ，すなわちワード単位文字データのオブジェクトに適用できるようになった．図 1 には文字データの APL オブジェクトの内部表現を示した.ただし変数名，定義関数名および作業空間名にはこのデータ・タイプは適用できない。この拡張において，PC-APL との互換性は維持される．

実際に拡張するにあたり，IBM Madrid Scientific Center で開発された IL(Intermediate Language) でインタープリタを記述し，それをコンパイルして 5550 の CPU である 8086のアセンブラのコードに落とした．

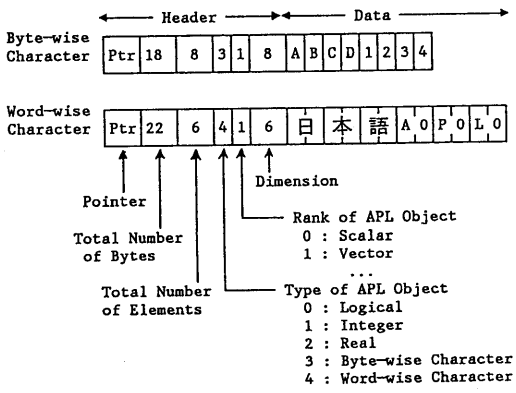

図 1．文字データの内部表現

エクゼキュータ

APLのエクゼヒュータは，インタープリタの日本語DOS およびマシンに依存する部分とのインタフェースの役割を担っている．その主な機能はキーボード，スクリーン，プリンタ等の入出力および記憶域およびファイルの管理である．以下に表示上必要となる APL 記号の問題とその入出力の管理方法について述べる．

1. APL記号とフォント

5550 の日本語 DOS は，シフト JIS のコード体系を採用しており，システムの入出力はこのコードに従う．既に 256のコード・ポイントは全て占有されており，もはや 60余個の APL 記号を配置する余裕はない．そこで筆者らは，1バイトの片仮名のコード・ポイントに APL記号を導入した (図 2 参照).APL と外の世界，例えば画面やファイル等の入出力は，このコードを通じて行なわれる．一方実際の APL のフォントは，5550 の場合 VRAM 上にフォント・エリアが取られているため，APL のロード時にファイルから適当なフォント・エリアにロードすることにより入出力できるようになる．一方原子ベクトルは PC-APL との互換性を持たせるため，その文字配置を図 3 のように採った．エクゼキュータは APL システムの内と外とのコード変換を行う．

2. キーボードと画面の管理

通常の DOS のモードでは，5550 のキーボードは半角 (1 バイト) の英数字,片仮名と全角 (2 バイト) の英数字，片仮名，平仮名および仮名漢字変換による漢字をサポートしている．この組合せの上に，APL記号を導入したために，実際のキーボードは APL ON/OFF の 2 つのモードを持っている (ただし日本語 APLでは半角の片仮名はコード・ポジション上存在しないため入力できない)．このモードの切り換えは，Ctrlキーと後退キーの組合せによって自由に行える。全角文字の入力はいずれのモードでも可能にしている．一方画面の管理はキーボードの操作に合わせて表示する文字の位置および種類を正しく認識し，1 行ごとの実行をできるようにしてある．

また日本字に対応するデータ・タイプを導入したため，1 バイト文字と 2 バイト文字が混在したデータを画面上に表示する場合，直接データを画面に出すと 1バイト文字の直後に x"00" いわゆる Null コードが現れ，ユーザから見れば余分な空白が生じる．そのため実際に表示する場合この Null コードを削除して画面に出している．ただしこのようなデータを行列として表示する場合には，行と列の関係を保つために下記のごとく Null コードを削除せずに表示している．

```
      3 3⍴'日本語APL'
日本語
A P L
日本語
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p				⍀	⍳	⍑		
1	SOH	DC1	!	1	A	Q	a	q			→	⍞	⍺	⍨		
2	STX	DC2	"	2	B	R	b	r			⍓	∈	⊂	\		
3	ETX	DC3	#	3	C	S	c	s			⍫	≤	⍵	⍋		
4	EOT	DC4	$	4	D	T	d	t			⍸	⍠	⌈	⊢		
5	ENQ	NAK	%	5	E	U	e	u			≡	≥	↓	≠		
6	ACK	SYN	&	6	F	V	f	v			⍲	~	⍳	∨		
7	BEL	ETB	'	7	G	W	g	w			⍱	∇	⍥	○		
8	BS	CAN	(	8	H	X	h	x			⍷	∆	⍈	⍁		
9	HT	EM	)	9	I	Y	i	y			⍋	⍈	⍂			
A	LF	SUB	*	:	J	Z	j	z			⍎	⊥	⍟			
B	VT	ESC	+	;	K	[	k	{			⍒	⊃	∪	≠		
C	FF	FS	,	<	L	¥	l	|			⍬	⍳	⍱	∧		
D	CR	GS	-	=	M	]	m	}			⍟	⍴	÷	↑		
E	SO	RS	.	>	N	^	n	¯			⍲	×	×	←		
F	SI	US	/	?	O	_	o				⍙	∩	∘	⍋		

図 2. 日本語 APL の入出力コード

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	*	⍋		C	S	h	x	.	≡					SYN	`
1	←	×	⍱	⊂	D	T	i	y	CR	⍂					SOH	@
2	→	!	⍒	⊃	E	U	j	z	LF	⍞					STX	"
3	↑	|	⍎	∘	F	V	k	⍙	BS	⍠					ETX	#
4	↓	⌈	⍬	⍺	G	W	l	_	SP	⍈					EOT	$
5	∈	⌊	⍈	⍵	H	X	m	¯	TAB	⍂					ENQ	%
6	⍳	=	⍨		I	Y	n	0	⍞	⍟					ACK	&
7	⍴	≠	⍫		J	Z	o	1	⍓	⍱					BEL	DC4
8	,	>	⍳	;	K	⍙	p	2	'	⊢					DLE	NAK
9	?	≥	⊥	]	L	a	q	3	⍲						DC1	ETB
A	~	<	⍑	[	M	b	r	4	∇						DC3	DC2
B	○	≤	≠	)	N	c	s	5	⍱	US					VT	{
C	+	⍲	⍲	(	O	d	t	6	⍠	*	∪				FF	|
D	-	⍱	/	:	P	e	u	7		¥				CAN	SO	}
E	÷	∧	\	A	Q	f	v	8	RS	^				EM	SI	DEL
F	⍟	∨	∩	B	R	g	w	9	⍸					SUB	FS	ESC

図 3．日本語 APL の原子ベクトル

補助プロセッサ

前で記したように，日本語 APL では日本字を 1 字を 1 要素として扱うために，実際のデータの長さと要素数とは必ずしも一致しない．例えば，

```
Dat←'日本語'
```

は APL のデータの要素数としては 3 であるが，物理的な長さは 6 バイトを占める．このようなデータをプリンタやファイルに出力する場合，実際のカラム数やバイト数がユーザとしては必要になってくる．つまりワード単位で表現されている文字データのバイト数を知るためには次のような関数をユーザが定義すればよい．

```
      ∇ Z ← LENGTH  T
[1]   Z ← (⍴T)++/256 < ⎕AV ⍳ T
[2]   ∇
```

この関数 LENGTH の変数 T に Dat を与えると 6 という値が返ってくる．しかしこの方法であると，大きいデータの場合には，パーフォーマンスの点で問題がある．筆者らは，より効率的にこのような変換を行う機能を補助プロセッサ AP555 で実現した．この補助プロセッサーは文字データに関して (1) ワード単位のデータ・タイプとバイト単位のデータ・タイプ間の相互変換，(2) 文字ベクトルのバイト長の計算および (3) シフト JIS コード上の連続コードと文字データとの相互変換の機能を持つ．

APL2 での日本語処理

日本語 APL の制約

今まで述べた日本語 APL はシフト JIS コードに依存した形で実現したため，他の 2 バイト文字を扱う国のコード言語（韓国語，中国語等）には直接適用できない問題をもつ．また，APL 記号を 1 バイトの片仮名のコードポジションに配置したため，そのデータは APL のオブジェクトとして扱うことができなかった．例えば中国語 (Simplified Chinese, Hanji) の場合は，1 バイトコードの x"81" から x"FC" に 2 バイト文字の第 1 バイトが存在するため，ASCII コード以外の 1 バイト文字を配置する場所は無くなる．従ってこのような言語をサポートするためには新たな方法を導入する必要がある．筆者らは日本語 APL のより一層の機能的強化を果たすため，APL2 を現在ワークステーション上に実現しようとしている．ここではこれらの問題を何のように解決するか，その試みについて簡単に述べてみる．

多国語処理への対応

APL2 の稼働環境として Operating System/2 (OS/2) をベースとする．OS/2 では NLS(National Language Support)機能としてコードページという概念を導入している．コードページはシステムに使われる文字セットを規定するものであり，OS/2 では動的に切り換えることが可能である．またシステムが DBCS Enviromental Vector を提供するため，それらの値を用いれば，自分がどのようなコード体系にあるのか判断できる．従って，2 バイト文字か 1 バイト文字かを判断する部分を特定のコード体系に依存しないプログラムを作れば，複数の 2 バイト文字に対応可能な処理系が実現できる．（但し現状ではハードウェアの制約上，同時に複数の多バイト文字をサポートできる訳ではない．）

APL2 の設計

APL2 では日本語 APL と同様に，英数字文字も日本字も 1 文字 1 要素として扱うことにした．日本語 APL では文字データのタイプとしてバイト単位文字とワード単位文字の 2 種類設け，APL 記号をバイト単位文字として扱ったが，これではどうしても扱うことのできない文字が生じる．そこで，APL2 のインタープリタ内部では 1 バイトの文字も 2 バイトとして表現するようにした．例えば 1 バイトの片仮名は対応するコード（1 バイト）と APL が内部的に定義したコード ID（1 バイト）の組になって扱われる．そしてシステムからの入力に関しては，エクゼキュータが 1 バイト文字をインタープリタの解釈できる形式に変換を行う．その逆に出力時には，システムが正しく処理できるようにデータを変換する．このようにすることで多少処理のオーバーヘッドが生じるが，APL のオブジェクトとして扱えるデータの範囲は広まり，また中国語のようなコード体系にも対応できるようになる．

結語

本論文では，パーソナルコンピュータの上で動く日本語 APL 言語システムを設計するにあたっての考察およびその実現方法について述べた．従来の APL との互換性を堅持しながら，日本字のデータを従来の英数字と同様に処理できるようにするには，筆者らの採用した方法が最も妥当であると思われる．ただし物理的な入出力操作において，今までの慣習上，データの"長さ"と"要素数"の違いに異和感を持たれる場合があるかも知れないが，文字のエンコーディング方法に対してまで，高級言語のユー

ザは気をつかう必要はないと筆者らは考えている．今後 “同時”多言語処理の要請が増せば，現在の日本字を2バイトで表現するだけでは限界があり，より多バイトの表現が求められてくると予想される．その際にも，データの内部表現を気にせずに扱えることが重要になると思われる．プログラミング言語の持つ意味，定義や仕様をできるだけ“自然な形”で拡張することが大切なのではないだろうか．

謝辞

APL インタープリタの開発に協力してくださった Dr. Manuel Alfonseca を中心とする IBM Madrid Scientific Center の諸氏に感謝いたします．

文献

[1] 宇土，秋元，金子，佐貫: パーソナルコンピュータ上の日本語 APL の試作．情報処理学会第 29 回全国大会,1D-5 (1984).

[2] M.L.Tavera, M.Alfonseca, J.Rojas: An APL system for the IBM Personal Computer. IBM System J. Vol.24, No.1 (1985).

[3] 宇土，秋元，金子，佐貫: 日本語 APL の漢字データ処理．情報処理学会第 31 回全国大会,9E-1 (1985).

[4] IBM Corp.: 日本語 APL 言語解説書. IBM Publication N: SB18-1094 (1985).

[5] 木下恂: 新しい日本字コード系による日本語処理の実現．情報処理学会マイクロコンピュータ研究会, 37-1 (1985).

[6] IBM Corp.: APL2 Language Reference Manual. IBM Publication SH20-9227-2 (1988).