IVERSON 言語 (A Programming Language)†

所　真理雄††

1. はじめに

プログラミング言語に対する一般的評価を求めることは極めて困難である．これはプログラミング言語のあり方についての基本的な考え方，接する立場，使用目的の相違から評価の基準が一定しないためである．Iverson 言語についていえば，この言語がオリジナルな形では実現(implement)できないこと，したがってオリジナルな言語と実現された言語のどちらについて議論しているのか明確でない場合が多いこと，またわが国では解説が十分に行なわれていないことなどもさまざまな評価を受ける原因となっていると思われる．

Iverson 言語は 1962 年 IBM Watson Research Center の Kenneth E. Iverson により開発された言語である．Iverson 自身はこれに APL (A Programming Language) という名称を与えているが，後に実現された APL\1130[11] や APL\360[12),13)] などとの混同を避けるために，本解説においてはこのオリジナルな言語のみを Iverson 言語と呼ぶことにする．単に APL と書かれた場合は Iverson 言語の実現された言語の総称を意味する．

Iverson 言語が発表された当時は，電子計算機は第二世代の最盛期であり，ソフトウェアの分野では FORTRAN II が広く数値計算に用いられ，ALGOL や COBOL が実現されようとしていた．またいくつかの非数値処理言語やシミュレーション言語も前後して現われた．このような電子計算機の商品化の時代に「ハードウェアからアプリケーションプログラムまでの共通の言語」というキャッチフレーズの下に Iverson 言語は発表されたのである．Iverson は開発にあたって，「電子計算機における情報の処理は実際の計算機の実行に沿って記述されたならば極めて複雑なものになってしまう．人間の理解は一連の処理をコンパクトにまた抽象的にとらえ，詳細をみないことによってなされる．各種プログラミング言語が，目的とする記述のレベルでのプログラミングのために開発されてきているが，1) 各種言語の習熟は時間と労力の無駄となる．2) 言語をはっきりレベルに分けることは困難である．例えば高いレベルの言語を用いている人がある部分については低いレベルの言語で記述したいことがある．3) ハードウェアとソフトウェアの明確な分離ができない場合がある．例えばマイクロプログラム．4) ハードウェア設計者とソフトウェア設計者の間の緊密な連絡のために同一言語が必要である[1),2),3)]」と述べている．そしてハードウェアの機能的記述 (functional description) からアプリケーションプログラムの記述までの多数の Iverson 言語による記述例が示された[1)~8)]．

Iverson 言語は Iverson 自身が名づけた A Programming Language という名前にもかかわらず，電子計算機のプログラミング言語というよりもアルゴリズム記述言語あるいはアルゴリズムの記法とみるのが妥当である．しかしながらベクトルやマトリックスの演算の表現が簡潔であるため，アレー処理用言語としての実現が試みられた[9),10)]．そして記号や表現方法において数々の変更や補足が行なわれ，1967 年に APL\360[12),13)] が稼動した．

本解説では先ず Iverson 言語を平易に説明し，その実現について述べる．最近 APL を用いたアルゴリズムの記述が数多くの論文でみられるようになってきているので[21)~14)]，その中心的存在である APL\360 についても簡単に説明する．

2. Iverson 言語

Iverson 言語は Iverson 記法と呼ばれることも多い．Iverson 言語は形式的に定義されておらず，本章ではその基本的な部分を A Programming Language[1)] に沿って解説する．

2.1 文法と意味

2.1.1 基本文法

Iverson 言語によるアルゴリズムの表現（プログラ

† Iverson's Language (A Programming Language), by Mario Tokoro (KEIO University)

†† 慶応義塾大学工学部

ム）は平面的に配置された文からなる．ここで Iverson 言語の基本文法を示してみよう．（ALGOL N 記法を用いる．）

〈文〉::=〈代入文〉|〈比較分岐文〉

〈代入文〉::=〈被演算作用素名〉←〈式〉

〔⟶〕

〈比較分岐文〉::=〈式〉：〈式〉$\overset{\langle 関係\rangle}{\longrightarrow}$

〈関係〉::=＝|≠|＞|≯|≥|≱|＜|≮|≤|≰|∈|∋

大きな矢印⟶は自分以外の文へと伸びているものである．代入文，比較分岐文ともに式がまずはじめに評価され，そのあと代入文ならば代入が行なわれる．代入文から出ている大きな矢印があるときは，すべての場合にこの矢印で示された文へと分岐する．比較分岐文ならば，：の両側の2つの式の値が大きな矢印の上に示された関係を満たす場合に限りその矢印で示される文へと分岐する．これ以外の場合は次の文へと移

表 1 Iverson 記法†

	識別子名	記法	定義 0—ORIGIN INDEX
被演算作用素名	スカラー	a	
	ベクトル	**a**	$\mathbf{a}\equiv \mathbf{a}_0,\ \mathbf{a}_1,\ \mathbf{a}_2,\cdots,\ \mathbf{a}_{(\rho\mathbf{a})-1}$
	マトリックス	A	$X\equiv \begin{bmatrix} A^0_0 \cdots\cdots A^0_{(\rho A)_1-1} \\ \vdots \\ A^{(\rho A)_0-1}_0 \cdots A^{(\rho A)_0-1}_{(\rho A)_1-1} \end{bmatrix}$ $\nu\mathbf{a}$：ベクトルの要素数 A^i：i 行 A_j：j 列 μA：行数 νA：列数
	配列名	記法	定義および例
特殊配列	Full Vector	$\mathbf{w}\leftarrow\varepsilon$ (n)	$\mathbf{w}_i\equiv 1$　$\varepsilon(5)\equiv(1,1,1,1,1)$ ε ≡zero vector
	Unit Vector	$\mathbf{w}\leftarrow\varepsilon^j$ (n)	$\mathbf{w}_i\equiv(i=j)$　i=j となる所だけ 1 $\varepsilon^0(5)\equiv(1,0,0,0,0)$ $\varepsilon^3(5)\equiv(0,0,0,1,0)$
	Prefix Vector	$\mathbf{w}\leftarrow\alpha^j$ (n)	$\mathbf{w}_i\equiv(i<j)$　j 以前が 1 $\varepsilon^3(5)\equiv(1,1,1,0,0)$
	Suffix Vector	$\mathbf{w}\leftarrow\omega^j$ (n)	$\mathbf{w}_i\equiv(i>n-j)$　後から j コが 1 $\omega^3(5)\equiv(0,0,1,1,1)$
	Infix Vector	$\mathbf{w}\leftarrow i\downarrow\alpha^j$ (n)	Rotation を見よ．i コの 0 のあとに j コの 1 $2\downarrow\alpha^4(6)\equiv(0,0,1,1,1,0)$
	Interval Vector	$\mathbf{k}\leftarrow\iota^j$ (n)	$\mathbf{k}_j\equiv i+j$　j からはじまる整数 $\iota^0(3)\equiv(0,1,2)$ $\iota^3(3)\equiv(3,4,5)$
	Full Matrix	$W\leftarrow E_{(m\times n)}$	$W^i_j\equiv 1$　すべて 1 のマトリックス
	Identity Matrix	$W\leftarrow I_{(m\times n)}$	$W^i_j\equiv(i=j)$　対角成分が 1
	Random	w ←?	0 か 1 のどちらかをランダムに取る
		w ←? (n)	各要素が 0 か 1 かの n 列のベクトル
		w ←?j (n)	$\mathbf{w}_i\equiv$0 か 1，$+/\mathbf{w}\equiv j$
	演算	記法	定義および例
算術および論理演算	算術	+, −, ×, ÷	普通の定義
	絶対値	z ← | x	
	FLOOR	k ← ⌊ x	$k<x<k+1$　⌊5.8≡5
	CEELING	k ← ⌈ x	$k-1<x<k$　⌈5.8≡6
	剰余	k ← m | n	$n\equiv(m\times q)+k,\ 0\leq k<m$　15 | 32≡2
	AND	w ← u ∧ v	
	OR	w ← u ∨ v	
	否定	w ← ũ	
		w ←∼ u	
	関係	w ← x ℛ y	w≡1　iff xℛy is true
	MAX	z ← x ⌈ y	
	MIN	z ← x ⌊ y	
	べき剰	z ← x ⁎ y	$z\equiv y^x$

	識別子名	記法	定義および例	
基底	Base k Value	$l \leftarrow k \perp m$	ベクトル m を k 進で読む	$10 \perp (3,2,1) \equiv 321$ $16 \perp \binom{1,0,0}{0,1,0} \equiv \binom{256}{16}$
	Base 2 Value	$l \leftarrow \perp u$	ベクトル u を 2 進で読む	$\perp (1,0,1,0) \equiv 10$
	Representation	$l \leftarrow k_{(n)} \top j$	$\nu l \equiv n,\ k \perp l \equiv k^n \mid j$	$10(3) \top 144 \equiv (1,4,4)$ $10(2) \top 144 \equiv (4,4)$
		$u \leftarrow (n) \top j$	$\nu u \equiv n,\ \perp u \equiv 2^n \mid j$	$10(4) \top 144 \equiv (0,1,4,4)$ $(4) \top 13 \equiv (1,1,0,1)$
選択	Compression	$c \leftarrow u/a$	$u_i = 0$ の a_i を取りのぞいた a	$(1,0,1)/(3,4,5) \equiv (3,5)$ $\alpha^1/a \equiv a_0, a_1, a_2$
	Row Compression	$C \leftarrow u/A$	$C^i \equiv u/A^i$	$(1,0,1)/\binom{0,1,1}{1,0,1} \equiv \binom{0,1}{1,1}$
	Column Compression	$C \leftarrow u \not/ A$	$C_j \equiv u/A_j$	$(1,0,1) \not/ \binom{0,1,1}{1,0,1}{1,1,1} \equiv \binom{0,1,1}{1,1,1}$
	Row List Compression	$c \leftarrow E/A$	$c \equiv A^0,\ A^1,\ A^2, \cdots, A^{(\nu A)-1}$	
	Expansion	$c \leftarrow u \backslash b$	$u/c = \bar{\varepsilon},\ u/c \equiv b$	$(1,0,1) \backslash (3,1) \equiv (3,0,1)$
	Row List Exapansion	$A \leftarrow E_{(m,n)} \backslash c$	$\mu A \equiv m,\ \nu A \equiv n,\ E/A \equiv c$	$E(2,3) \backslash (1,2,3,4,5,6) \equiv (1,2,3,4,5,6)$
	Mask	$c \leftarrow /a : u : b/$	$\hat{u}/c \equiv \hat{u}/a,\ u/c \equiv u/b$	$/(1,2,3) : (1,0,1) : (9,9,9)/ \equiv (9,2,9)$
	Mesh	$c \leftarrow \backslash a : u : b \backslash$	$\hat{u}/c \equiv a,\ u/c \equiv b$	$\backslash (8,9) : (1,0,1,0,1) : (3,2,1) \backslash \equiv (3,8,2,9,1)$
シフト	Left Rotation	$c \leftarrow k \uparrow a$	$c_i \equiv a_j,\ j \equiv (\nu a) \mid i+k$	$3 \uparrow (1,2,3,4) \equiv (4,1,2,3)$
	Right Rotation	$c \leftarrow k \downarrow a$	$c_i \equiv a_j,\ j \equiv (\nu a) \mid i-k$	$3 \downarrow (1,2,3,4) \equiv (2,3,4,1)$
	Left Shift	$c \leftarrow k \Uparrow a$	$c \equiv \bar{\alpha}^k \wedge k \uparrow a$	$2 \Uparrow (1,2,3,4) \equiv (3,4,0,0)$
	Right Shift	$c \leftarrow k \Downarrow a$	$c \equiv \alpha^k / k \downarrow a$	$2 \Downarrow (1,2,3,4) \equiv (0,0,1,2)$
縮約	Reduction	$c \leftarrow \odot / a$	$c \equiv a_0 \odot a_1 \odot a_2 \cdots \odot a_{(\nu a)-1}$	$+/x \equiv x_0 + x_1 + \cdots x_{(\nu x)-1}$
	Row Reduction	$c \leftarrow \odot / A$	$c_i \equiv \odot / A^i$	$+/\binom{1,2,3}{4,5,6} \equiv (6,15)$
	Column Reduction	$c \leftarrow \odot \not/ A$	$c_j \equiv j\odot / A_j$	$+ \not/ \binom{1,2,2}{4,5,6} \equiv (5,7,9)$
	Matrix Product	$C \leftarrow A \overset{\odot_1}{\odot_2} B$	$C^i_j \equiv / \odot_1 A^i \odot_2 B_j$	$A^{+}_{\times} B$ は普通のマトリクス積，⊙は二項演算子または二項関係
その他	Catenation	$c \leftarrow a, b$	$c \equiv a_0,\ a_1, \cdots, a_{(\nu a)-1},\ b_0,\ b_1, \cdots, b_{(\nu b)-1}$	
	Row Catenation	$C \leftarrow a \oplus b$	$C_0 \equiv a,\ C_1 \equiv b$	$\mathcal{C}^0(3) \oplus \mathcal{C}^1(3) \oplus \mathcal{C}^2(3) \equiv \binom{0,3,5}{1,4,6}{2,5,7}$
	Indexing	$c \leftarrow a^m$	$c_i \equiv a_{m_i}$	$(3,2,1)_{(1,0,1)} \equiv (2,3,2)$
		$C \leftarrow A^m$	$C_i \equiv A_{m_i}$	
		$C \leftarrow A^m$	$C^i \equiv A^{m_i}$	
	Maximum Prefix	$w \leftarrow \alpha^i / u$	$w \equiv \alpha^i(\nu u)$，j は $\wedge/w/u \equiv 1$ であるものの最大のもの	$\alpha/(1,1,0,1,0,0) \equiv (1,1,0,0,0,0)$
	Maximum Suffix	$w \leftarrow \omega^i / u$	$w \equiv \omega^i(\nu u)$，j は $\wedge/w/u \equiv 1$ であるものの最大のもの	$\omega/(1,0,1,1,0,1) \equiv (0,0,0,0,0,1)$
	Representation	$w \leftarrow \rho(x)$	w は文字 x の 0,1 コード表現	

† 文献 1), 2), 3), 4) より一部翻訳して転載.

る．分岐をこのように表わすことにより文にラベルを付ける必要がなくなった．

2.1.2 被演算作用素 (Operand)

Iverson 言語は被演算作用素としてスカラ，ベクトル，マトリックスを取り扱う．(トリーも扱うことができるがここでは省略する．) この 3 種類の被演算作用素名としてアルファベットの全文字について，小文字 (lower case)，小太文字 (boldface lower case)，大文字 (upper case) がそれぞれ割当てられる．これにより被演算作用素の内包と外延を合わせて表現できる．

被演算作用素名のディメンジョンおよびその保持する変数の形の宣言はない．本解説においては各字体で表わされる被演算作用素名について，u, v, w は論理値，h, i, j, k は整数値，x, y, z は実数値，a, b, c は任意の形の値を保持するものとする．ベクトルおよびマトリックスの始点，すなわち添字が何からはじまるかはあらかじめ任意に指定することができる．ここでは 0 を始点 (0-origin index) として指定したものとする．

2.1.3 特殊配例 (Special Array)

Iverson 言語の特徴の 1 つは特殊なベクトル，マトリックスを表わす記号が備わっていることである．これを単独に，または演算子と組合わせることによりアレー処理を極めて表現することができる．

2.1.4 演算子 (Operator)

演算子には単項演算子，二項演算子，三項演算子がある．多くの単項演算子はあいまいさを導びくことなく二項演算子としても用いられる．二項演算子の左側の被演算作用素は制御作用素 (Controller) とも呼ばれる．演算子間に優先順位はなく，演算は右から左へと一意的に実行され値を返す．実行の順序を変更するには括弧（ ）が用いられる．算術および論理演算子はスカラだけでなく形の同じベクトル間またはマトリックス間においても要素ごとに適用される．スカラとベクトルまたはスカラとマトリックス間の演算は，形が等しく全要素がそのスカラと同じ値を持つベクトルまたはマトリックスとみなして適用される．さらに列数の等しいベクトル**a**とマトリックスAとの演算も，各列のすべての要素が**a**のその列の要素と同じ値を持つようなAと形の等しいマトリックスであるとみなして適用される．

Iverson 言語は演算子の数が多いことでも有名であるが，中でも特徴のあるものを簡潔に説明する．**基底** (Base Value) とは行ベクトルの各要素を制御作用素で示された基底の各桁として，これを十進数に変換する演算である．⊥**l** は 2⊥**l** の省略形である．制御作用素はベクトルであってもよく，(24, 60, 60)⊥(3, 15, 20)≡(((3×60)+15)×60)+20．**基底に基づく表現** (Representation) はちょうどこの逆の演算である．**選択** (Selection) には論理アレーである制御作用素の値が1である場所に対応する被演算作用素の要素を取り出す**圧縮** (Compression) やその逆である**拡張** (Expnsion) などがある．**縮約** (Reduction) は演算子とベクトルまたはマトリックスの形をした被演算作用素との間で定義され，被演算作用素の要素間にその演算子で示される演算を適用することを意味する．これをもとに，**一般化されたマトリックス積** (Generalized Matrix Product) が定義されている．この他にも Iverson 言語独特の演算子として**連結** (Catenation)，**インデックス付け** (Indexing)，**最大接頭辞** (Maximum Prefix)，**最大接尾辞** (Maximum Suffix) などがある．

2.1.5 サブルーチン，入出力

サブルーチンの実際的な定義の方法については仕様が定まっていない．入出力については磁気テープファイルについてのみ述べられているが，あまり実際的記述には向かないようである．

2.2 プログラム例

プログラム 1-(a) はある計算機の instruction fetch の基本部分の記述である．メモリーM，プログラムカウンタレジスタ**s**，命令レジスタ**c**，アドレスレジスタ**a**，バッファレジスタ**b**に対して instruction fetch とは M の ⊥**s**（論理ベクトル**s**を二進数として読んだ値）行目の内容を**c**に入れること，すなわち **c**←M⊥**s** であるが，これは直接できない．先ず**s**の内容を**a**に転送し，メモリアクセスを行なうと M⊥**a** の内容は**b**にはいる．この時 M⊥**a** の内容は消えるので再書込みを行ない，**b**の内容を**c**に転送して instruction fetch を終了する．レジスタ間の転送が OR(∨) で行なわれるので転送先を前もってリセット（ε は全要素がゼロのベクトル）しておかねばならないことから，これは（a）のようになる．

プログラム 1†

（a）

```
──→ a ← ε
    a ← s ∨ a
    b ← ε
    b ← M⊥a ∨ b  ┐
    M⊥a ← ε      ┘
    M⊥a ← b ∨ M⊥a
    c ← ε
    c ← b ∨ c ──→
```

（b） 1-ORIGIN INDEX

```
──→ l ← ε(0)
    s ← ε(0)
                         =
──→ ν(c): 0 ──────────────────→
    x ← c₁
 =  c ← ᾱ¹/c
┌── x : (
│                        =
│   x : ) ─────────────────────┐
│                        ∈     │
│   x : p ───────────────┐     │
├── l ← x ⊕ l            │     │
├── s ← x ⊕ s       ←────┘     │
│   l ← s₁ ⊕ l        ←────────┘
└── s ← ᾱ¹/s
```

（b）は完全括弧式（すべての演算に順序を示す括弧がついている式）をポーランド記法になおすアルゴリズムを示している．**c**に入力文，例えば〔(A+B)×C〕があるとし，**l**を答がはいるベクトル，**s**を演算記号の一時スタック，**p**を演算子の集合を表わすベクトルであるとする．このプログラムの実行の結果**l**は ×C+BA となる．

2.3 ま と め

以上 Iverson 言語の本質的な部分であって，さらに部分集合として意味のあるものを取り出して解説し

† 文献3より転載．

た．この言語が計算機上での実行を意識していないアルゴリズム記述言語であるということは，サブルーチンや入出力の仕様がほとんど決まっていないことによってもわかる．Iverson は Iverson 言語をハードウェアからソフトウェアまでの共通の言語であると強調したが，彼自身統一的に「記述」可能であるという意味で主張したと思われる．統一的に記述可能であるという反面には，プログラマは対象をどのレベルで記述しているのかを明確にしておく必要がある．

一方，独特な被演算作用素や特殊配列の表現，基底，選択　シフト，縮約という極めて有効な演算などによって，論理処理，文字処理，アレー処理のアルゴリズムが簡潔に記述されることが理解される．演算子の数が多く，その機能を他の演算子で表現できるものがあることや，特殊文字を多く使うことについて Iverson は，「機能がだぶっていても矛盾がなければ演算子の数は多いほど便利だ．覚え切れなければ気にいったものだけを使えばよい．特殊文字が多いのは，例えば FORTRAN では IF, DO, GOTO, MOD, MAX, MIN などの機能語があるのと同じで，特殊文字を用いた方が簡潔である[5]」と述べている．

Iverson 言語はいろいろな種類のアルゴリズムの記述に用いられたが，特に IBM システム 360 の形式的記述[6] をはじめとするハードウェアの機能的記述に大いに利用された．また並列処理の記述や，機能的記述とその機能を実現するハードウェアとの対応の記述について補ない，計算機設計言語として実現されたものもあり[15),16)]，ほとんどの計算機設計言語に影響を与えたといってもよいであろう[14)]．

3. Iverson 言語の実現

3.1 概　説

Iverson 言語の実現には本質的な困難は伴わない．構文解析は右から左へ一意的であり，パラメータの受け渡しについてもむずかしいことは何も要求されていない．1つの演算子がスカラ，ベクトル，マトリックスに用いられる点についても，いくつかの一般化された要素の走査ルーチンを準備して，走査ルーチンが各演算ルーチンを呼べばよいことが Iverson によっても述べられている[5)]．それにもかかわらず実現がやや遅れたのは Iverson 言語が自由すぎたためであり，1) 分岐の表現，2) 特殊文字，3) サブルーチンおよび入出力の表現，4) 実現の方式（どの程度のサブセットとするか，コンパイラかインタプリタか，オンラインかオフラインか）などを決定しなければならなかったためであろうと思われる．

実現は主に IBM の手によってなされた．1964 年 H. Hellerman は IBM 1620 にコンソールタイプライタからのオンラインシステム PAT[9)] を実現した．これはアレー計算を目的とした実験システムであり，上記の 4 点についての解決はみられない．次いで 1967 年に APL\360 が K. E. Iverson, A. D. Falkoff, L. M. Breed, P. S Abrams, R. D. Moore, R. H. Lathwell らの協力の下に完成した．これは IBM システム 360-50 に実現された約 100 端末を持ったオンラインインタプリタシステムで，専用のキーボードを用いて特殊文字の問題を解決し，分岐，入出力　サブルーチンの表現の仕様も解決している．現在 IBM はプログラムプロダクトとしてシステム 360-40 以上，IBM 1130, IBM 1050 に APL を供給しており，また Marquard 社，Scientific Time Sharing 社，APL Computing Service 社，Computer Innovation 社，APL Manhattan 社などが商用サービスを行なっているようである．

IBM 以外のメーカーは現在までこの言語の実現については正式な表明を行なっていないが，Burroughs や XDS には実現の動きがあるようであり，また CDC が STAR に Iverson 言語に似た言語を実現するであろうという噂もある．IBM 以外の計算機上で現在稼動しているものとして，Univ. of Massachusetts の CDC 3600, Univ. of Maryland の UNIVAC 1108, Univ. of Washington の Burroughs B5500, Montana State Univ. および Canadian Defense Research Establishment の XDS Sigma 7 などのシステムがあり，これらは学内または研究所内で大いに利用されているようである．このようなことからも IBM 以外の各社も近いうちに APL を供給するのではないかと思われる．APL のユーザー団体である SHARE APL は APL の標準化をはじめねばならない時期にきていると判断しているようである．

3.2 節で Iverson 言語の実現に先駆的役割を果し，今後もその中心的存在となるであろう APL\360 について簡単に紹介する．

3.2 APL\360

3.2.1 システムの概要

APL\360 は JOSS[26),27)] とほぼ同じレベルのシステムであり，端末から会話的にプログラムやデータを入力し，解釈実行する．（最近バッチ処理も可能となっ

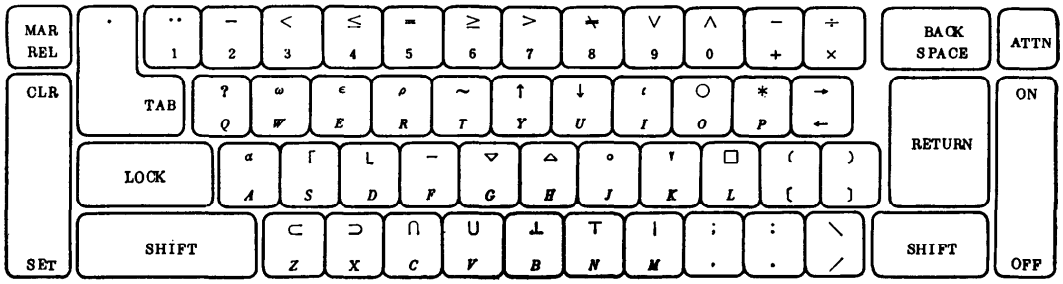

図 1 APL\360 KEYBOARD

た模様である.) 文法的エラーやステップごとの答も即時に得られ，プログラムは保存できる．しかしながらオブジェクトプログラムは得られないため他の言語で書かれたプログラムとのリンクはできない．端末はいわゆるテレタイプやカードリーダ／パンチ，紙テープリーダ／パンチが用いられ，電話回線を介してセントラルコンピュータと接続される．APL\360 は特殊なキーボードを用いる．この開発が現在の APL\360 をあらしめたといっても過言ではないであろう．これを図1に示す．

プログラムは1つの**ワークスペース**内で実行される．実行可能なワークスペースをアクティブな状態にあるといい，この時セントラルコンピュータのワーキングストレージの1つのブロックを占有している．このブロックのサイズは固定である．ワークスペースの一部はシステムの内部的 job のために使われ，残りの部分はステートインディケータ，シンボルディクショナリ，ソースプログラム，ブロック化されたオブジェクトプログラムが占める．ワークスペースはライブラリに保存することができる．**ライブラリ**にはプライベートとパブリックの2つがあり，プライベートライブラリは個人専用でロックすることによって機密が保持される．パブリックライブラリはだれでも使用できる．しかしながらそこにストアした人しか変更や削除はできない．

APL\360 は2つの言語——**システムコマンド**と**APL\360 オペレーション言語**とを持つ．APL\360 オペレーション言語は2つのモード——**実行モード**と**定義モード**を持ち，前者はプログラムの実行，後者はサブルーチン(ここではファンクションと呼ばれる)の定義を行なう．システムコマンドはオペレーション言語と機能的，形式的に異なり，これは，1) 端末制御コマンド（ワークセッションの開始，終了），2) ワークスペース制御コマンド，3) ライブラリ制御コマンド，4) インクワイアリコマンド（リストの打出し要求など），5) コミュニケーションコマンド（端末間の連絡）に分類される．形式的にはシステムコマンドは右括弧）ではじまる．すべてのシステムコマンドは実行モードで使用可能であり，定義モードでは使用が制限される．

3.2.2 APL\360 オペレーション言語

APL\360 オペレーション言語は Iverson 言語のサブセットであり，トリーの演算および三項演算子を含んでいない．またアレーの肩や足下に書かれていた添字はタイプライタにこの能力がないために変更され，演算子は1ペースに納まるように，また Iverson 言語とのコンパティビリティのために多くの二重印字が用いられる．またファンクションの定義や分岐の表現が確立した．これらはオンライン言語であるための特徴を持っている．

プログラムは基本的には式または代入文の列からなり，式に対してはその値が計算されて即座にタイプライタに印字される．式または文の終了は復帰改行によって行なう．式の実行は右から左に行なわれ，この変更は括弧（ ）を用いる．

変数名の形式はスカラおよびアレー（多次元アレーも可能）に共通で，英字ではじまる 77 文字以内の英数字列が有効である．英字とはA……Zおよびアンダーラインされた A……Z である．実数値は小数点およびまたはE（FORTRAN と同じ）によって表わされ，**文字定数**は ' で囲む．

ファンクションの定義は次のように行なう．先ず▽を打ち，そのあとに function の形および名前を入力し，復帰改行を行なう．するとシステムは定義モードになり，システムが文番号を打つ．そうしたら代入文または分岐文を打ち復帰改行を行ない，これをつづけ，最後に▽を打つと実行モードに戻り定義は終了する．**分岐文**は定義モードでのみ可能であり，分岐は→

プログラム 2†

```
      A←3
      B←4
      ((A*2)+B*2)*÷2          √A²+B²
5                             答
      ∇X←SUM Y; I             ファンクションの定義
[1]   X←0
[2]   I←0
[3]   X←X+I
[4]   I←I+1
[5]   →3×⍳I≤Y                I≦Y が真なら 3×⍳1 は 3,
                              偽なら 3×⍳0 は空
[6]   ∇                       ファンクションの定義終了
      I←20
      P←5
      Q←SUM P                 ファンクションの実行
      Q
15                            答
      I
20
      ∇TRIANGLE; S            ファンクションの定義
[1]   □←'INPUT A B C'        リテラルの出力
[2]   A←□                     入力の要求
[3]   □←'THE AREA IS'        リテラルの出力
[4]   S←0.5×+/A               (1/2)(A₁+A₂+A₃)
[5]   □←(S××/S-A)*0.5         √S(S−A₁)(S−A₂)(S−A₃)
                              の値を出力
[6]   ∇                       ファンクションの定義の終了
      TRIANGLE                ファンクションの実行
INPUT A B C
      3 4 5                   A₁=3, A₂=4, A₃=5 を
                              入力
THE AREA IS
6                             答
      A←3 4 ⍴⍳12              Aに1から12までの整数を
                              3行4列の形で代入する
      A
1  2  3  4
5  6  7  8
9 10 11 12
      ⍴A                      Aのデイメンジョンは
3 4                           3行4列である
      A[1 2; 2 3 4]           Aの第1行と第3行でかつ
                              第2列，第3列，第4列の
2  3  4                      ものは
10 11 12
      Q←A[1;], A[3]           QにAの第1行目と第3行
                              目を連結して代入する
      P←2 4⍴Q                 Qを2行4列の形にしてP
                              に代入する
      P                       Pは
1  2  3  4
9 10 11 12
      +/[1]P                  Pの要素の値を列毎に加え
                              ると
10 12 14 16
      I←0 1 0 1
      I/P                     IによるPの圧縮
 2  4
10 12
      P[; 1 2]+.×P[; 3 4]     Pの第1列および第2列か
                              らなるマトリックスと，P
 25  28                       の第3列および第4列から
137 156                       なるマトリクスのマトリク
                              ス積
```

† 文献 12 および 13 より一部転載.

表 2　APL\360 演算子†

（表 1 と異なるもののみ挙げる.）

算術および論理演算

×Y　((Y>0)−(Y<0))
÷Y　1÷Y
X*Y　X の Y 乗
*Y　e の Y 乗
X⍟Y　log_X Y
⍟Y　log_e Y
○Y　πY
X ○ Y　三角関数，逆三角関数††
X ! Y　Y から X を取る組合せ ₓC_Y
Y!　Y の階乗，Y-1 のガンマ関数.
?Y　⍳Y の要素のいずれかをランダムに取る.
X⍱Y　X NOR Y
X⍲Y　X NAND Y

一般化されたマトリックス演算

X⍟₁.⍟₁ Y　一般化された内積.
X⍟₁.⍟₂ Y　一般化された外積.

縮約

⌿／Y　Y の最後のディメンジョンに沿っての縮約.
⌿／[Z]Y　Y の Z 番目のディメンジョンに沿っての縮約.

圧縮と拡張

X／Y　Y の最後のディメンジョンに沿っての圧縮.
X／[Z]Y　Y の Z 番目のディメンジョンに沿っての圧縮.
X＼Y　Y の最後のディメンジョンに沿っての拡張.
X＼[Z]Y　Y の Z 番目のディメンジョンに沿っての拡張.

その他の演算

XρY　Y をディメンジョン X を持つよう並びかえる.
ρY　Y のディメンジョン.
X[Y]　X の Y 番目の要素.
X⍳Y　Y は X の何番目の要素であるか.
⍳Y　添字の始点から Y コの連続した整数.
X?Y　⍳Y の要素のいずれかを X コランダムに取る.
X⌽Y　Y の最後のディメンジョに沿って X だけ左へローテーション.
X⌽[Z]Y　Y の Z 番目のディメンジョンに沿って X だけ左へローテーション.
⌽Y　Y の最後のディメンジョンに沿って逆にする.
⌽[Z]Y　Y の Z 番目のディメンジョンに沿って逆にする.
X⍉Y　Y のディメンジョンを X で示されるように交代させる.
⍉Y　Y の転置行列.（Y の最後の 2 つのディメンジョンを交代させる.）
,Y　Y を 1 次元ベクトルにする.
X↑Y　Y の最初の X コを取る.
X↓Y　Y の最初の X コをすてる.
⍋X　X の要素の小ささの順序を示す.
⍒X　X の要素の大きさの順序を示す.

演算以外の記号

□←X　X の値を打出す.
X←□　数値の入力を要求する. X は入力された値を保持する.
X←⍞　リテラルの入力を要求する. X は入力されたリテラルを保持する.
'XYZ'　リテラル XYZ.

† 文献 12 および 13 より一部翻訳して転載.

††

(−A)○B	A	A○B
(1−B*2)*.5	0	(1−B*2)*.5
Arcsin B	1	Sine B
Arccos B	2	Cosine B
Arctan B	3	Tangent B
(¯1+B*2)*.5	4	(1+B*2)*.5
Arcsinh B	5	Sinh B
Arccosh B	6	Cosh B
Arctanh B	7	Tanh B

の右側の式の値を示す文番号またはラベルの付いた文へと行なわれる．その式に関係（満たされるとき1，そうでないとき0）を含めることにより条件分岐が記述される．→XにおいてXが0のときはその function から出ること，またXが空であるときは次の文へ移ることを示す．ファクションの定義の際に**局所変数**の指定をすることができる．これはファンクション名のあとにセミコロン；を打ち，そのあとに変数名をセミコロンで区切って打てばよい．このように指定されない変数名はすべて全体的である．ファンクションの**パラメータの受け渡し**はすべて call by address の形でおこなわれ，定義の形は零項演算子形，単項演算子形，二項演算子形およびそれぞれに答のパラメータを加えたもの，すなわち，FUNC, FUNC X, X FUNC Y, および X←FUNC, X←FUNC Y, X←Y FUNC Z, の形に限られる．

入出力には□と⍞（□と ' の二重印字）を用い，前者は数値，後者は文字の入出力を示す．これによってインタラクティブなプログラムを容易に作ることができる．

アレーの演算についてはディメンジョンや添字の表現方法が変更されている．ρA はAのディメンジョンを示す演算で BρA はAをBというディメンジョンを持つように並びかえる演算である．また A[B] は AのBで示される要素を取り出すことを指示する．圧縮や縮約のディメンジョンを示すのにもこの［ ］が用いられ，/[K]A は A の第K番目のディメンジョンについて，/A は A の最後のディメンジョンについて圧縮や縮約の演算を行なうことを示す．また一般化されたマトリックス積 $A\overset{\odot_1}{\underset{\odot_2}{.}}B$ は $A\odot_1.\odot_2 B$ に変更されている．

3.3 まとめ

APL\360 の長所としては次の点が挙げられる．1) プログラミングが容易である，2) ターンアラウンドタイムが短い，3) エラーメッセージが適切である，4) システムとしての使用が簡単である．一方短所としては，1) 実行速度が遅い，2) ファンクションが値を受け渡さない，3) アレーの要素がアレーではいけない，4) 他の言語で書かれたプログラムとリンクできない，などの点が挙げられる．その他特徴として，1) 演算子が豊富である，2) ブロック構造がない，3) データタイプの宣言がない，なども挙げられる．これらの長所，欠点，特徴は APL\360 がオンラインインタプリタシステムとして実現されたことに大きく依存している．

APL\360 は実行時間が短いことよりもプログラミングに要する時間ないしはターンアラウンドタイムが短いことを望むユーザに向いているシステムであるといえよう．現在 APL\360 は初学者を含む極めて幅広い層の人々によって，数値計算，事務計算，統計解析などをはじめとする極めて広い分野で利用されている．また，各種のシステム設計やコンパイラの作成などにおけるアルゴリズムの妥当性のチェックやアルゴリズムのドキュメンテーションとしても広く用いられるようになってきている．

APL\360 で行なわれたような Iverson 言語の実現方法とは全く別の方向からの実現のための研究もされている．アレー演算には並列処理が望まれるが，これをセルラマシンとして実現しようとする方向である[25]．また近年小型計算機やミニコンピュータでマイクロプログラム制御方式のものが出現しているが，これを利用した実現も有効であると思われる．

4. おわりに

以上，Iverson 言語およびその実現について述べてきたがここで簡単に振り返ってみる．

Iverson 言語は一種のアルゴリズムの記法であり，直接計算機での実行と結びつけて考えるのは妥当でない．この言語の特徴は多数の演算子を備え，アレー処理のアルゴリズムを簡潔に表現できることが第1に挙げられ，ハードウェアの機能的記述や多くの計算機設計システムの入力言語として応用されている．

Iverson 言語の実現は主に IBM によってなされ，APL\360，APL\1130 などがリリーズされている．これらはオンラインインタプリタシステムでプログラミングが容易である一方，実行時間や記述能力の点で不満も多い．しかしながら商用オンラインシステムとして稼動し，多数のユーザーを得ているという事実は評価されるべきものである．最近，わが国においても APL\360 が供給されるようになった．今後のわが国における反応に注目したい．

今後の APL の姿は IBM の政策によるところが大きいと思われるが，これとは別に，他の実現方法やアレープロセッサの入力言語への応用なども考えられてよいと思われる．

参考文献

1) K. E. Iverson, A Programming Language,

Wiley, 1962.
2) K. E. Iverson, A Programming Language, SJCC, 1962.
3) K. E. Iverson, A Common Language for Hardware, Software and Application FJCC, 1962.
4) K. E. Iverson, A Program Notation in System Design, IBM System J. June 1963.
5) K. E. Iverson, Formalism in Programming Language, CACM, Vol. 7, No. 2, 1964.
6) A. D. Falkoff, K. E. Iverson, E. H. Sussenguth, A Formal Description of System/360, IBM System J. Vol. 3, No. 3, 1964.
7) F. P. Brooks, K. E Iverson, Automatic Data Processing, Wiley, 1963.
8) A. D. Falkoff, Algorithm for Parallel Search Memories, JACM, Oct. 1962.
9) H. Hellerman, Experimental Persnalized Array Translator System, CACM, Vol. 7, No. 7, 1964.
10) P. S. Abrams, An Interpreter for 'Iverson Notation', Comp. Sci. Dept. Stanford Univ. Tech. Rep. CS-47, Aug. 1966.
11) P. C. Berry, APL\1130 Primer, IBM Corp. 1968 (C-20-1967-0).
12) APL\360 User's Manual, IBM Corp. 1968 (H 20-0689-0).
13) APL\360 Primer IBM Corp. 1968 (H 20-0689-0).
14) M. A. Breuer, General Survey of Design Automation of Digital Computers, Proc. IEEE, Vol. 54, No. 12, 1966.
15) H. P. Schlaeppi, A Formal Language for Describing Machine Logic, Timing, and Sequence (LOTIS), IEEE Trans. EC, Vol. EC-13, No. 8, 1964.
16) T. D. Friedman, S. C. Yang, Method Used in an Automatic Logic Design Generator, IEEE Trans. C, Vol. C-18, No. 7, 1969.
17) G. H. Foster, APL: A Perspicuous Language, Comp. and Automation, Nov. 1969.
18) H. Katzan, Jr., A Prose Glossary of APL (A Programming Language), Comp. and Automation, Aug. 1970.
19) D. D. McCracken, Whither APL?, Datamation, Sep. 1970.
20) J. P. Roth, W. G. Bouricius, P. R. Schneider, Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits, IEEE Trans. EC, Vol. EC-16, No. 5, 1967.
21) L. J. Woodrum, Internal Sorting with Minimal Comparing, IBM System J., No. 3, 1969.
22) H. G. Kolski, Problem Formulation Using APL, IBM System J., No. 3, 1969.
23) L. J. Woodrum, A Model of Floating Buffering, IBM System J., No. 2, 1970.
24) W. H. E. Day, Compiler Assignment of Data Items to Registers, IBM System J., No. 4, 1970.
25) K. J. Thurber, J. W. Myrna, System Design of a Cellular APL Computer, IEEE Trans. C, Vol. C-19, No. 4, 1970.
26) J. E. Sammet, Programming Languages: History and Fundamentals, Prentice Hall, 1969.
27) J. C. Shaw, JOSS: A Designer's View of an Experimental On-Line Computing System, FJCC, 1964.

(昭和47年7月6日受付)