会話型プログラミング・システム†

前島　亨††　下村建之††

1. はじめに

最近，人間とコンピュータとがコミニュケーションするための種々の端末装置が開発され，TSS で情報処理をする方向が活発になってきた．また，日本でも TSS のサービスが幾つか開始され，大型汎用コンピュータを日常の道具として利用する方向にある．これは，システムの高度化，複雑化に伴う問題の解決手段と共に誰もが必要なときにどこからでもコンピュータを使えるという面が強調される．この意味から，コンピュータが一般社会の道具として普及するためには，人間とコンピュータとの会話が充分うまくなされなければならない．

一般に，会話型情報処理システムとは，人間とコンピュータとの会話的相互作用によって問題を解決（情報処理）するために構成されたシステムといえる．ここでは，情報処理を限定した会話型プログラミングシステムについて考えてみる．これは，新しい問題の発生に対してその解法を見い出し，それをプログラミングおよび実行し，その結果によって解法の修正，別の解法の検討，パラメタの変更等を繰り返し，最適な解を得るといった一連の動作が，人間とコンピュータとの会話的相互作用により遂行できるシステムを意味する．ここで重要なことは，コンピュータとの応答が人間の思考過程の連続的推移の中で遂行され，また，人間の心理作用の許容範囲内で動作することにある．さらに，コマンドやプログラミング言語が充分使い易いものであり，端末機器が問題解決に適したものでなければならない．

本文では，言語を中心に会話型プログラミングシステムについて考察する．

2. 会話型プログラミング・システム出現の経緯

先ず，コンピュータの利用形態の面から会話型プログラミング・システム出現の経緯を概観してみる．

コンピュータが出現してから 1950 年代の中頃までは，人間がコンピュータの言葉（機械語）でプログラムを記述し，それをコンソールのブレイク・ポイントや表示ランプをセットしながら処理していた．この使い方は本質的にはオンライン方式であるが，コンピュータを１人で占有し，人間の思考や動作時間とコンピュータの処理速度の差異から利用効率に問題があった．そこで人間をできるだけコンピュータから隔離するバッチ処理方式に発展した．この方式では，中央処理装置の処理速度と，入出力装置等の機械的動作速度との差が問題となり，機械的入出力処理を分離したサテライト・コンピュータ・システムが考えられた．さらに，一台のコンピュータでデータ処理と入出力操作とが非同期的に処理できるハードウェア機構が開発されバッチ処理効率を大幅に向上させた．1960 年頃には，多重プログラミング方式が開発され，スーパーバイザの管理により複数個のプログラムを並行処理することによりシステムのリソースの効率使用が計られた．

このような処理方式の発展経緯は，コンピュータの使用効率を高めることに主眼が置かれ，人間はできるだけコンピュータから遠ざけようとする方向にあった．一方，処理要求の急激な増加により，ターン・アラウンド・タイムが数時間から数日を要するという事態になり，プログラムやデータのわずかの誤りでも，その影響は非常に大きいものとなった．ハードウェアの進歩とオペレーティング・システムの開発によりトータルのスループットは格段に向上したが，処理をかかえた個々人の立場で考えた非効率さの問題が残された．

かかる背景から，再度，人間とコンピュータとを結

† Conversational Programming System, by Toru Maejima and Takeyuki Shimomura (Central Research Lab., Nippon Electric Co., Ltd.)

†† 日本電気株式会社中央研究所

びつけるシステムとした提唱されたのが TSS である．TSS は，人間とコンピュータとの会話的相互作用による問題解決能力の向上と，システムの経済的使用との課題を同時に解決したものといえる．米国においては 1961 年の終りに MIT 計算センタで最初の TSS[20]を開発し，その後，研究機関や大学等で幾つかの実験システム，教育用システムが開発され，1964 年頃から商用システムが稼動開始した．

TSS と関連した概念として，オンライン・リアルタイム・システムとリモート・ジョブ・エントリとがある．前者は特定業務の処理を指向した専用システムで，後者はセンタのバッチ処理の入出力機器が遠隔に設置され，実行はバッチ処理とほぼ同じであるので，ここでは特にふれない．

最近，ミニコンピュータの普及により，限定された分野あるいは小規模の問題解決に会話モードを取り入れ，ミニコンピュータを孤立した1つのシステムとして使用する形態も存在する．

3. 会話型言語の処理方式と基本機能

会話型プログラミング・システムは，一般に大型で汎用の TSS システムの上で実現している．ここで，システムの会話性について考えてみる．誰もが，任意の時間に，キーボードやディスプレイ等の端末機器を使ってシステムをアクセスでき，システムからは，それに対する応答を速やかに端末に返すという手続きの継続的な実行可能性である．この場合，システムのリソースが充分活用され得なければならない．この人間とシステムとのインタフェイスとして，コマンドおよびプログラミング言語があり，また，会話性を高めるための処理方式の検討が必要である．

問題解決のプロセスにおける会話性は，次の三つのステップで考えられる．

① プログラム作成段階の会話．
② デバッギングおよびテスト段階の会話．
③ プログラムの実行段階の会話．

会話性を導入することにより，①はプログラムの作成変更等を容易にし，②はコマンドを用いた動的デバッギングを可能にし，必要なら①へ戻り，③は実行中に端末からデータを入力，あるいは，パラメタの変更を可能にし，必要なら①または②へコントロールを戻す．これらの会話性は，システムのリソースや，端末機器の他に，プログラミング言語の処理方式に大きく依存する．ここでは，

① コンパイラ方式，
② インタプリタ方式，
③ インクレメンタル・コンパイラ方式[25),30)]

について，その処理方式と特徴を表1にまとめた．この内容は一般的なもので，実際にはインプリメンタにより相違がある．

次に，プログラミングによる問題解決の手順を，インタプリタ方式の処理例として図1に示す．

ここで，会話型プログラミングシステムにおけるコマンド，言語およびシステムの基本的機能をまとめてみる．

先ず，コマンドはシステムに対してサービスを要求するための言語と考えられ，したがって，コマンドの種類が豊富であればそれだけシステムのサービス機能

表1　処理方式および特徴

	コンパイラ方式	インタプリタ方式	インクレメンタル・コンパイラ方式
処理方式	・プログラムをすべて入力後コンパイラの1フェーズとして構文解析を行う． ・プログラムを一括してオブジェクトコードを生成する． ・オブジェクトコードを連続的に実行する．	・1ステートメント入力毎に構文解析を行う． ・1ステートメント入力毎に中間レベルの言語に変換する． ・中間レベルの言語を解釈しながら連続的またはステートメント毎に実行する．	・1ステートメント入力毎に構文解析を行う． ・1ステートメント入力毎にオブジェクトコードを生成する． ・オブジェクトコードを連続的または1ステートメント毎に実行する．
特徴	・プログラムをすべて入力後一括して処理するために会話性が乏しい． ・ステートメントの修正毎にプログラム全体の再コンパイルが必要． ・連続実行のため実行時の会話性がない． ・実行効率が良い． ・メモリ効率が良い．	・1ステートメント毎に処理するのでエラー修正が容易． ・修正したステートメントだけ中間レベルの言語に変換すれば良い． ・1ステートメント毎または部分実行が可能で変数の値の表示，代入等実行時の会話が可能． ・ダイナミック・デバッギングが容易． ・解釈実行のため効率はやや悪い． ・メモリ効率はやや悪い． ・直接実行文が使える．	・1ステートメント毎に処理するのでエラー修正が容易． ・修正したステートメントだけコンパイルすれば良い． ・1ステートメント毎または部分実行が可能． ・ダイナミックデバッギングが可能． ・実行効率はやや悪い． ・メモリ効率（特にコンパイル時）が悪い． ・直接実行文が使える．

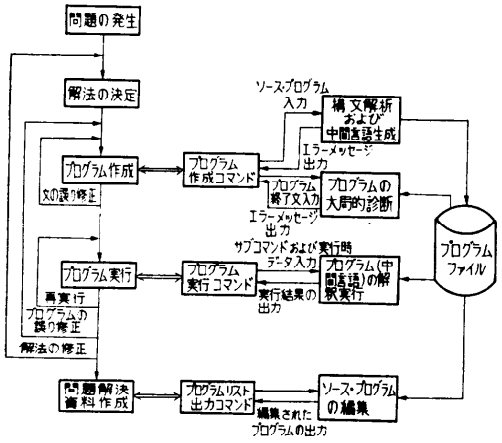

図1　インタプリタ方式による問題解決手順

が完備しているといえる．コマンドを機能別に分類すると次のものがある．

① システムとの会話開始および終了コマンド．
② プログラムの作成，変更，リストの出力に関するコマンド．
③ プログラム・ファイルやデータ・ファイルを扱うコマンド．
④ プログラムの実行制御用コマンド．
⑤ ファイルやシステムの情報の問合せに関するコマンド．
⑥ その他（特殊問題処理用コマンド等）．

望ましいコマンドおよびそのプロセッサの条件として次のものがあげられる．

① コマンドが覚えやすく，適当な省略形が使える．
② コマンドの種類が豊富で，利用者の作成したコマンドも使える．
③ コマンドのパラメタに省略時の自動解釈がある．
④ コマンドの誤り等のメッセージは即時応答がなされ，利用者により簡略形と詳細形の選択が可能．
⑤ コマンドのカタログ化が可能．

次に，プログラミング言語およびそのプロセッサについて考えてみる．現在，TSS で使われている言語の中に，バッチ処理用言語として開発されるものを改良したものが多くある．このことは，ほとんどすべての言語が，TSS の上でコマンドを使って会話型言語として使用可能といえる．しかし，人間の思考を直接端末機器を介してシステムに入力したり，手近の端末から，誰もが自由にシステムを利用可能にするためには，次のような考慮が必要である．

① 言語は，専門プログラマ以外の人でも理解しやすく覚えやすい．
② 種々の省略時の解釈が設定され，詳細な指示がなくても標準形の処理がなされる．
③ 卓上計算型言語のように，入力と同時に実行する直接ステートメントが使える．
④ デバッギング用ステートメントが使える．
⑤ バッチ処理の同一言語との互換性がある．
⑥ プログラムの１ステートメント入力毎にエラー診断がなされ，即時に応答する．
⑦ 種々の応答メッセージは，利用者によりその内容の詳細レベルを選択できる．
⑧ 端末からの入力形式はフォーマット・フリーである．
⑨ コマンドとの関連において，次のようなダイナミック・デバッギングが可能である．
- プログラムの実行の中断，再開．
- 連続的実行とステップ実行モードの指定．
- 任意の変数の値の参照，変更．
- プログラムの部分実行．
- 実行のトレース．

⑩ フォーマット・フリーで入力したプログラムを読み易い形式に編集して出力する．

なお，具体的言語機能については，もちろんその用途によって検討されるべきものである．

最後に，一般的にシステムの要件については次のものがある．

① 応答時間が速い．
② 同時に複数の利用者にサービスができ，各々が互いに独立に処理できる．
③ ファイル処理機能を備えている．
④ プログラム・ライブラリが完備している．
⑤ 処理を中断しても，そこまでの状態がファイルに保存され，後で継続処理が可能．
⑥ 端末機器の操作ミスが処理に直接影響を与えない．
⑦ 端末機器が人間工学的に配慮され，会話モードの処理に有効である．
⑧ プログラムやデータ・ファイルの共同利用および機密保持機能がある．

4. 会話型プログラミング・システムの例

国外で開発された会話型プログラミング・システム

の主なものについて，その目的およびシステム構成等を概説する[19]．

（1） JOSS

JOSS[1)~4)](JONIAC **O**pen-**S**hop **S**ystem)は RAND 社で JONIAC コンピュータ用に開発された実験的な TSS 用システムで 1964 年初めに稼動開始した．その後，1965 年に PDP-6 用に改良された．

このシステムの目的は，科学者や技術者がコンピュータの操作やコンパイラ，デバッギング方式等を学び小さい問題を自分で処理することができるように設計されたものである．ステートメントを入力後，直ちに実行する直接ステートメントとプログラムとして格納する間接ステートメントとがあり，小型の数値計算を主としている．卓上計算機的機能もある．

JOSS から派生した言語には，CAL, AID, TELCOMP, CITRAN 等がある．

（2） BASIC

次章で詳しく述べる．

（3） QUIKTRAN

QUIKTRAN[21),22)]は，IBM 社で IBM 7040/44 用に開発したオンライン・システムで，1963 年中頃に稼動開始した．

当初の目標は，FORTRAN プログラムの誤り除去機能を強力にすることであった．直接ステートメントと間接ステートメントとがある．既存の FORTRAN との互換性が保たれているので，FORTRAN プログラムの生産性を高めることに寄与した．

（4） RUSH, CPS

RUSH (**R**emote **U**sers of **S**hared **H**ardware) は Allen-Babcock 社と IBM 社が共同開発した小さな PL/I サブセットで会話モードに拡張したシステムである．IBM で使われているシステムは，CPS (**C**onversational **P**rogramming **S**ystem) と呼ばれ，1966 年秋から IBM 360/50 で稼動している．

このシステムの目標は，利用者にとって，できるだけ簡潔で使いやすく，かつ，PL/I の構文をできるだけ多く含むことにあった．JOSS 等と同様に直接ステートメントと間接ステートメントが使える．

（5） MATHLAB

MATHLAB[23)](**Math**ematical **Lab**oratory) は，シンボリック・コンピュテーションの機能をもつ最初のオンライン・システムで，SDC において AN/FSQ-32 TSS, MITRE において IBM 7030, MIT の MAC 計画において CTSS で稼動している．

このシステムの目的は，数学者またはその分野の人人に使いやすいコンピュータを計算の道具として提供することにあった．例えば，式や方程式の加法，置換，微分，積分，ラプラス変換，行列の乗算等の機能を提供した．入出力は，タイプライタまたは固定した文字表示装置をもつキーボードが利用された．

（6） APL/360

次章で詳しく述べる．

（7） AMTRAN

AMTRAN[26)] (**A**utomatic **M**athematical **Trans**lation) は，NASA によって IBM 1620 用に開発されたインタプリティブ・システムである．

システムの基本的目標は，非専門プログラマでも数学的問題を容易に解決できるオンラインシステムを提供することにあった．入出力は，キーボードや図形表示装置が利用できる．現在は，AMTRAN 72[39)] に改良されている．

（8） DIALOG

DIALOG[28)]はイリノイ工科大学で開発され，UNIVAC 1105 TSS のもとで 1966 年初めから使用されている．

このシステムは，図形表示装置を用いて，図形や文字情報の表示および必要なハード・コピーができる．使用に際しては特別な訓練を必要としないように配慮されている．行番号を与えるか否かで，間接ステートメントと直接ステートメントを区別して使うことができる．

以上の外に，会話型プログラミングシステムとして MAP[27)] (**M**athematical **A**nalysis **P**rogram), Culler-Fried システム，SPEAKEASY 等があり，また，音声による応答を試みたシステムとして，コーネル大学で開発された Tele-CUPL, Audio-CUPL[38)] がある．

5. BASIC と APL

ここでは実例として BASIC と APL について概説する．BASIC は TSS 用に設計され，現在最も広く使用されている代表的な会話型システムとして，APL は非常に特色ある言語であり，代表的な会話型システムとはいい難いが言語面からも会話型システムという観点からも，将来への示唆に富んだシステムとしてとりあげた．

4.1 BASIC[5),6),7),8),10)]

BASIC (**B**eginner's **A**ll-purpose **S**ymbolic **Instru**-

ction Code) は 1965 年, Dartmouth 大学に於て J. Kemeny, T. Kurtz の指導のもとに GE 225 用として開発された. その名の示すように, 初心者にとって使い易く, また熟練者にも便利なシステムとの定評がある. 主として科学技術計算に使用されているが, 改良を加えた事務計算用の BASIC も開発されている. BASIC は現在多くの TSS や小型計算機にインプリメントされている. ここではそれらの基本的な機能について述べる.

(1) BASIC 言語

(a) 文字集合と定数

英字, 数字の他に +−*/↑=><",.( ); $? が使用される. 数値定数は整数, 小数, E 形式が許され, 文字列定数はその両端を引用符号 " で囲む. 但し, 文字集合, 数値の精度等はシステムにより幾分異る点がある.

(b) 数値

変数名は英字1文字または英字1文字に数字1文字を続けた2文字で表す. 配列 (ベクトルとマトリックス) 名は英字1文字, 文字列変数及び文字列配列名は数値変数の表現に $ 記号をつけた形で表す.

(c) システム関数

関数名は全て3文字で, 例えば, SQR, SIN, COS TAN, ATN ($\tan^{-1}$), LOG, EXP, ABS (絶対値), INT (整数化), RND (乱数発生) 等がある.

(d) 基本的な文

全ての文にはその先頭に行番号がつけられる. この行番号が実行順序を示し, 飛び先指定 (GOTO 文等) に対するレーベルにもなる. また後に述べるがプログラムの文単位の修正, 追加等にも使用されている. 主な文を表2に示す.

IF 文は比較が正しければ指定した行番号に飛び, 正しくなければ次の文が実行される. FOR 文と NEXT 文によりループを表現する. NEXT の後には, 対応する FOR 文と同じ制御変数が指定されていなければならない. INPUT 文はプログラム実行時の端末装置からのデータ入力文である. READ 文は DATA 文で定義されたデータを順に変数に読み込む. PRINT 文は実行時の端末装置へのデータ出力である. その出力形式はシステムによって定められている. サブルーチン呼び出しには GOSUB 文を用いる. BASIC のサブルーチンの入口はプログラム中で固定されたものではなく, 行番号 (GOSUB 文で指定された) で示される. サブルーチンの終りは RETURN 文で示される. この戻り点も, プログラム中でスタティックに決るのではなく, 実行時に GOSUB 文のあと一番最初に実行された RETURN 文が戻り点になる. DEF 文では関数を定義することができる. 但し関数名は FN に英字を1字つけた形 (FNA〜FNZ) でなければならない.

表2 BASIC の基本文

LET <変数>=<式>
IF <式><比較演算子><式>THEN<行番号>
GOTO <行番号>
FOR <添字なし変数>=<式>TO<式>[STEP<式>]
NEXT <添字なし変数>
INPUT <変数> [, <変数>]…
READ <変数> [, <変数>]…
DATA <定数> [, <定数>]…
PRINT <式> [, <式>]…
STOP
END
REM <コメント>
GOSUB <行番号>
RETURN
DEF FN<英数>(<パラメータリスト>)=<式>

(e) その他の文

(d)であげたもの以外に, データファイルを扱う文, ベクトルやマトリックスを宣言する DIM 文, ベクトルやマトリックス演算, 入出力を行う MAT 文や MATREAD 文, MATPRINT 文がある.

例えば
```
MAT C=A*B
MAT PRINT C
```
では, 行列AとBの積を求めCに代入し, 行列Cを特定の形式で出力することができる.

このほか, ON 文による計算形 GOTO や多重代入文, 出力書式の指定等のできる文がある.

(2) BASIC 言語システム

言語システムとして使い易いように, 種々のコマンドが用意されている. コマンドの種類はシステムによって異るが, いずれも会話性を高めるために,

(i) プログラムファイルの保存, 共用,
(ii) プログラム1文入力毎の構文チェック,
(iii) プログラムの文単位の変更, 追加, 削除,
(iv) プログラムの即時実行 (Compile Link and GO),
(v) プログラムの部分実行,
(vi) ステップ実行,
(vii) 実行時のデバッグ機能 (変数の値の参照・変更),
(viii) 他のプログラムとのリンク, 呼び出し,
(ix) ソースプログラムの標準形式への編集

等のことが行なわれている．また処理形式も，コンパイラ，インクリメンタルコンパイラ，インタプリタ方式等種々ある．

（3） BASIC の特色

（i） 簡単な英語や数式で文を作り，文法が簡単．

（ii） 入出力の標準形式があり，初心者にもわかりやすい．

（iii） 数値計算用のシステム関数が豊富．

（iv） 行列演算用の文がある．

この言語の特色に加えて，TSS のもつ機能が（2）のように生かされている．

4.2 APL[12),13),14),15),16)]

APL (A Programming Language) は Iverson により 1962 年開発され，その設計目標はアルゴリズムを簡潔明確に記述することであった．その特徴は基本的な演算を定め，それをシステマティックに拡張し，ベクトルやマトリックスを扱えるようにした点にある．この結果，記述能力は高まり，ハードウェア，マイクロプログラム，システムプログラム，応用プログラム等の多方面のアルゴリズムを簡潔に記述できることが知られている．

しかしながら，この言語をプログラミング言語として用いるには，幾つかの問題点がある．APL では特殊な記号を含む大きな文字集合が前提であるが，Iverson 自身ものべているように，これらは現在の多くの入出力装置には入っていない[12)]．更に，記述能力に比して，プログラミングが容易か否か，プログラム自体のもつ文書性，プログラムデバッグの難易度等が問題になろう．

このような議論は，APL がプログラミング言語として有用なのか，あるいはハードウェアから応用プログラムに亘る広範囲なアルゴリズムの共通な記述法として有用なのかという問題に発展する[19)]．

ここで紹介する APL/360 は，APL のサブセットのオンライン解釈実行システムである．IBM 社 Watson Research Center に於て，A. Falkoff, K. Iverson により開発されたものである．使用者は特殊な文字をもつキィボードから，センタの APL/360 を使用できる．APL 言語の記述性と TSS の会話性を組合せた会話型システムである．

（1） APL/360 言語

（a） 文字集合

大文字の英字，数字のほかに次の 52 文字がある．

＋－×÷⍟＜≤＝≥＞≠⍒⍋．：，；¨（）〔〕？⍕－∘∧∨←↓→↑＼／⊤⊥∩∪⊃⊂⌈⌊～△▽○□⍺⍵∈⍴⍳

このほか上の文字を組合せた合成文字，例えば⍲⍱⌽等を使用する．キィボードでは重ね打ちすることで実現される．

（b） データ

APL で扱われるデータは全て，タイプ（数値，文字，論理値）と構造（スケーラ，ベクトル，マトリックス等）を持っている．数値定数は数字と．E ¯の13文字で表現される．

（c） 演算子と文

演算子は右被演算子を1つもつ単項演算子が左右両方に被演算子をもつ2項演算子のいずれかである．

APL の文とは，演算子と被演算子の列である．演算子間に優先順位はなく，文の実行の際は右から左へ演算子が順次実行され，先に実行された演算結果が次の演算子の右被演算子となる．例えば 3×5+1 の結果は 18 となる．特に順序を指定するには括弧を用い

APL の演算子はタイプと構造をもつデータを被演る．

算子とし，算術演算のように定まったタイプでの演算（構造面での拡張がある），異った構造をもつデータを生成する演算（（e）参照），それらの混合演算（（e）参照）等を行うものといえる．

表3に基本的な演算子を示している．同じ演算記号が左被演算子を持つか否かによって，単項，または2項演算子となる．またこれらは，ベクトルやマトリックスの演算に拡張される．

（d） 変数

英字，数字および△よりなる文字列で変数名を表

表3　APL の基本演算子

演算子	単項演算	2項演算	演算子	単項演算	2項演算
＋	何もせず	加算	－	符号をかえる	減算
×	Signum	乗算	÷	逆数	除算
⍟	Exp	べき	⌈	cealing	最大値
⌊	floor（ガウス）	最小値	∣	絶対値	余り (mod)
⍟	自然対数	底指定の対数	○	π 倍	円関数
！	階乗	組合せ	？	乱数	ランダムな組合せ
＜		Less than	≤		Less than or equal
＝		equal	＞		Greater than
≥		Greater than or equal	≠		not equal
∧		and	∨		or
⍲		nor	⍱		nand
～	not		←		値の代入

す．変数にはグローバルなものと，定義関数（(ⅱ)参照）内だけで有効なローカルなものである．

変数の値となるデータのタイプや構造の宣言は行わない．それらは実行時の値指定(specification)演算子の実行により決定される．したがって，変数は特定の値を保持する計算機内の記憶領域と考えるより，むしろ数値定数や定数ベクトルあるいは演算の結果作られたデータ（スケーラであれベクトル，マトリックスであれ構造とタイプをもつ）につけられた名前と考える方がわかりやすい．

（e） ベクトル及びマトリックス

数値を要素としてもつベクトル（定数ベクトル）は数値をブランクで区切って表現する．定数や定数ベクトルに種々の演算子を作用させて，更に大きなベクトル，マトリックス，3次元以上の配列を構成することができる．

スケーラの場合と同様に，変数名を指定して値を格納し後の時点で使用することもできる．また配列の要素あるいは部分配列をとり出すには角カッコ［ ］を用いる．

APL では文字列定数を引用符号'で囲んで表現するが，文字列データは，各要素が文字よりなるベクトルとして扱われる．従って［ ］により部分文字列等が扱える．

ベクトルやマトリックスを被演算子とする演算は豊富にあるが次のように分類できる．

（ⅰ） element by element

基本演算子の被演算子が配列のとき，対応する要素毎の演算で定義する．

（ⅱ） Reduction

今，かりに◎を任意の基本演算子とし，×をベクトル長さn）とするとき

◎／× は ×[1]◎×[2]◎…◎×[n] を意味する．

（ⅲ） マトリックス演算

+.×は通常のマトリックスの乗算を行う．これを拡張して generalized inner product, outer product がある．

（ⅳ） Compression と Expansion

／はベクトル（配列）の縮小，／は拡大（数値なら0，文字列ならブランクをつめる）を行う．

（ⅴ） その他の演算

自然数よりなるベクトルの作成，ベクトルの連結，2次元以上の配列の作成，部分配列をとる，配列内要素の置換や巡回置換，値の換算，要素を大きさの順に並べる，配列のベクトル化，逆行列を求める，行列の割算（逆行列を乗ずる）等複雑な機能（定義は明確だが）をもつ演算子がある．

例えば $4\times\left(\frac{1}{1}-\frac{1}{2^2}+\frac{1}{3^2}-\cdots-\frac{1}{10^2}\right)$ の計算は，APL では 4×－／÷(⍳10)*2 と表現される．但し，－／は Reduction，⍳10 は1から 10 までの自然数よりなるベクトル1　2…10 を作る．÷は単項演算子で逆数，この場合各要素の逆数をとる．

（f） 定義関数と実行モード

APL／360 には実行モードと定義モードがあり，前者では，端末からの入力1文が即時実行され，計算結果が印字される．但し最後の演算子が値の代入のときは印字されない．定義モードでは，入力文の実行はなされず，何行にもわたる関数を定義することができる．定義された関数は，その名前が実行モードで使用されるか，または実行されている関数の中で使用されていると関数呼び出しとなり，実行される．両モードの変更は端末から▽記号を入力することによって行われる．

関数は定義時に，(ⅰ)関数名，(ⅱ)演算結果を返すか否か，(ⅲ)アーギュメントの個数(0,1 または2個)が決められる．結果の値を返す関数は，文中で演算子のように使用できる．特にアーギュメントをもつものは単項あるいは項演算子のように使える．アーギュメントの個数は増大2個だが，ベクトルや配列をアーギュメントとすることにより実質的には何個でもパラメータをもってよいことになる．

関数内での変数はグローバルなものと，関数内だけで有効なローカルなものとがある．ローカルな変数は関数の定義時に宣言されなければならない((2参照)．

定義関数内の各文には行番号がつき，文の実行順序を示す．実行順序の制御には演算子→と行番号を用いる．

定義関数の実行時のデータの入出力には記号□や・を使って，文中にかける．

（2） APL／360 システム

APL 言語をサポートしているシステムについて述べる．

まず，重要な概念として Work space がある．これは，端末ユーザに対して割りあてられる一定のセンタ計算機の記憶領域である．この中には，実行に必要な作業領域，変数，定義関数，端末に関する情報等が

入っている．ユーザは Work space 単位で情報を保存，共用が行える．言語の中でグローバルな変数というのは1つの Work space で有効なという意味である．

言語の周囲にいろいろなコマンドが用意されているが，

（i） 端末装置の制御，

（ii） Work space の管理，

（iii） Work space の保存，呼び出し，

（iv） システムの情報を問合せ，

（v） 他の端末とのコミュニケーション

等を行うものがある．

会話型システムで重要なのは，早く正しい結果を出すことだが，プログラムの診断，修正が容易に行えなければならない．このために，定義関数のリストアップ，文単位の修正等に多くの配慮がなされている．文の診断については，いくつかの簡単なエラーメッセージがある．例えば，被演算子の性質が演算子に合わないとか，変数が連続しているとかである．APL 言語の演算子は機能が大きいため，使用者の意図とは異っていても，言語的には正しいということがある．この結果プログラムの言語上のエラーは非常に簡単なメッセージとなっている．

（3） APL 言語の特徴

（i） 文は演算子と被演算子の列（式）で構成される．従って文のシンタックスは簡単である．

（ii） データはタイプと構造をもつ．構造として2次元以上の配列も使える．

（iii） 変数とはデータに付けられた名前と考えられ，その値のタイプや構造の宣言は行わないで，実行時の値の代入で決定される．

（iv） 演算子の数が多く，その機能が大きい．

（v） 演算子を次のように機能分類できる．

① 定まったタイプをもつデータの演算．

② 新しい構造をもつデータの作成．

③ ①と②の混合．

④ その他（飛び起し→や□等）．

（vi） 演算記号を組合せて，機能的にまとまった1つの演算子となる．

（vii） 演算子がどういう演算を行うかは，被演算子となるデータのタイプと構造により決定される．この点他の手続き言語を異る．

（viii） 関数定義ができ，関数名を演算子のように使える．

6. 日本における会話型システム

わが国においても，1960 年の後半より TSS が開発され，多くのシステムが稼動している（表4参照）．それに伴って，端末装置を利用する会話型言語が開発されて来た[40]．

これまで開発された会話型言語は，（i）従来の高級言語を会話型に改造したもの，（ii）会話型言語として設計されたもの，（iii）特殊目的の言語，（iv）カルキュレータ型に分類できる．ここでは（i）～（iii）についてのべる．

FORTRAN は 1967 年に慶応大学において TOSBAC 3400/30 で開発され，その後 NEAC-TSS，電電公社の DEMOS-FORTRAN 等がある．DEMOS

表4 日本における TSS 稼動状況

（TSS の普及と通信回線問題：中嶋栄之助 Computer Report, 1971. 7 増刊号 (p. 63) 参照）

名　称	設置場所	稼動	機　種	言　語
FACOM-ETSS	富士通	1966	FACOM 230/50	ALGOL
KEIO-TOSBAC	慶　大	1967	TOSBAC 3400/30	FORTRAN
阪大 MAC	阪　大	1968	NEAC 2200/500	FORTRAN BASIC ADAPT
ETSS	電総研	1968	HITAC 8400	ASSEMBLER BACIC
5020 TSS	日　立	1968	HITAC 5020	PL/IW
DIPS-O	NTT	1968	HITAC 8400	PL/I
京大 TSS	京　大	1969	FACOM 230/60	BACCUS LISP
東北大 TSS	東北大	1969	NEAC 2200/500	FORTRAN BASIC
日電 TSS	日本電気	1969	NEAC 2200/500	FORTRAN BASIC COBOL ASSEMBLER
トヨタ自動車 TSS	トヨタ自工	1970	UNIVAC 1108	ALGOL FORTRAN
ユニバック TSS	日本ユニバック他	1970	UNIVAC 1108	ALGOL FORTRAN
九大 TSS	九　大	1970	FACOM 230/60	FORTRAN
通産省 TSS	通産省	1971	NEAC 2200/500	FOTRAN BASIC COBOL
DEMOS	NTT	1971	J 3050	FORTRAN
通電 TSS	電　通	1971	GE 235	ALGOL BASIC FORTRAN
CALL-360	日本 IBM	1971	IBM 360	BASIC PL/I FORTRAN

においては，バッチとの互換性が考慮され，また1行ずつのシンタックスチェックを行なっている[18]．この他には，日本ユニバックの CALG (Conversational ALGOL), NEAC-TSS COBOL, FACOM 230/60 用の CPL (Conversational PL/I, 日本情報処理開発センタ）等がある．

（ii） のタイプとしては，BASIC が多くのシステムで開始されている．ETSS-BASIC(電総研), HITAC 5020 TSS BASIC があり，5020 TSS ではインクリメンタルコンパイラ方式がとられている．さらに，HITAC 8700 用 BASIC, FACOM 23045 S/55 用 BASIC, NEAC-TSS BASIC が開発され，NEAC-TSS では，1行入力毎のチェックの他に，実行時のデバッグ機能がある[10]．BASIC 以外に，JOSS の方言である BACCUS (Basic Calculus), CONCISE (Conversational Computationand Interactive Simulation Execution) が各々富士通(株)，日立製作所により開発されている．

(iii)の特別な応用向きとして，LISP (京大 TSS), ADAPT (阪大 TSS) がある．また電電公社のプッシュホンによる計算システム DIALS も会話型システムにとりあげられる．

これら大型 TSS によるもののほかに，弧立システムとしてミニコンピュータを使用した会話型システムが数多くある．その中には FORTRAN や BASIC 言語をもつものもある．

7. おわりに

会話型プログラミング・システムとして今後解決されなければならない問題点は幾つか存在するが，特に言語について若干ふれてみる．

現在使用されている言語にはバッチ処理系の汎用言語を改良したものが多い．これは，バッチ処理で習得した言語をそのまま使えるとか，互換性を保ち，デバッギングに使える等，充分利用値価がある．しかし，今後は，誰もがコンピュータを利用できる状況を考えると，会話性を考慮した使い易い言語の開発が必要となる．さらに，言語の標準化の検討もすべきであろう．

一方，コンピュータ・アプリケーションの多様化に伴い，従来の汎用言語の他に会話型の問題向き言語の開発も志向されなければならない．

これらの言語は，日本人が使う以上は日本語の自然言語を主体としたプログラミング言語は如何にあるべきかという問題が生ずる．これには，カナ文字の他に漢字を含めた場合の日本語の扱いの困難さから，入出力端末機器を含めた広範囲の研究が必要となる．

自然言語によるコミュニケーションには，音声を含めた本来の意味での会話が望まれる．先に述べた電電公社のプッシュホンによる DIALS は，音声による応答を実現しているし，またコーネル大学の実験もあり，将来は問題向きの会話システムとして期待できる．

以上，言語を中心として会話型プログラミング・システムの現状及び問題点を述べた．コンピュータとの会話は，人間がコンピュータと接触している時間が長くなるので，人間の心理的要因を含めた人間工学的諸問題が十分に解決されていく必要がある．

参 考 文 献

1) S. L. Marks: THE JOSS PRIMER, RAND Corp.
2) G. E. Bryan, J. W. Smith: JOSS LANGUAGE, RAND Corp.
3) E. P. Gimbel: PROBLEM-SOLVING WITH JOSS, RAND Corp.
4) J. C. Shaw: JOSS: A Designer's View of an Experimental On-line Computing System, Proc. FJCC, p. 455, (1964).
5) J. G. Kemeny, T. E. Kurtz: BASIC, Dartmouth, 大, (1968).
6) T. E. Kurtz et al.: The Dartmauth Time-Sharing Computing System, Dartmouth.
7) J. G. Kemeny, T. E. Kurtz: BASIC PROGRAMMING, John Wiley, 1967 (森口他訳: ベーシック入門, 共立出版.)
8) BASIC Language Ref. Manual, 電通.
9) CALL/360 BASIC Manual, IBM
10) NEAC-TSS BASIC 会話型言語説明書, 日本電気.
11) P. C. Dressen: The data BASIC Language—A data processing Language for non-professional programmers, Proc. STCC, p. 307, (1970).
12) K. E. Iverson: A Programning Language, Proc. SJCC, pp. 345～351 (1962).
13) K. E. Iverson: A Programming Langnage, John Wiley, (1962).
14) H. Katzan Jr.; APL Programming and Computer Techniques, Van Nostrand, (1970).
15) APL/360 Primer, IBM マニュアル.
16) L. Gilman, A. J. Rose: APL/360 An Interactive Approach, John Wiley, (1970)
17) 電話計算 (DIALS) 説明書, 日本電電公社.
18) DEMOS エディタ及び構文チェッカ説明書,

日本電電公社.
19) J. E. Sammet: Programming Languages: History and Fundamentals, Prentice-Hall, (1969). (竹下亨訳: プログラミング言語ハンドブック, 日本経営出版会.)
20) F. J. Corbato et al.: An Experimental Time-Sharing System, Proc. SJCC, Vol. 21, p. 335 (1964).
21) T. M. Dunn. J. H. Morrissey: Remote Computing: An Experimental System Part 1: Experimental Specifications, Proc. SJCC, Vol. 25, p. 413, (1964).
22) J. M. Keller, E. C. Strum, G. H. Yang: Remote Computing: An Experimental System, Part 2: Internal Design, Proc. SJCC. Vol. 25 p. 425, (1964).
23) C. Engelman: MATHLAB: A Program for On-Line Machine Assistance in Symbolic Computations, Proc. FJCC. Vol. 27, p. 117, (1965).
24) K. Lock: Structuring Programs For Multi-program Time-Sharing On-line Applications, Proc. FJCC. pp. 457～472, (1965).
25) J. L. Ryan et al.: A Conversational System For Incremental Compilation and Execution in a Time-Sharing Environment, Proc. FJCC, pp. 1～21, (1965).
26) J. Reinfelds et al.: AMTRAN A Remote-Terminal, Conversational-Mode Computer System, ACM 21 st Nat. Conf. p. 469 (1966).
27) R. Keplow et al.: Man-Machine Comunication in On-Line Mathematical Analysis, Proc. FJCC. Vol. 29, p. 465. (1966).
28) S. H. Cameron et al.: DIALOG: A Conversational Programming System with a Graphical Orientation, Com. ACM. Vol. 10, No. 6, p. 349, (1967).
29) E. J. Sandewall: LISPA: A LISP Like System for Incremental Computing. Proc. SJCC. (1968).
30) M. Peccoud et al.: Incremental Interactive Compilation, IFIP. pp. 89～97, (1968).
31) G. Y. Bretbard: The ACME Compiler, IFIP pp. 38～44, (1968).
32) J. I. Schwartz: Interactive Systems: Promise Present and Future, Proc. FJCC. pp. 89～97, (1968).
33) R. K. Moore: Interactive Languages: design criteria and a proposal, Proc. FJCC. pp. 193～218, (1968).
34) H. Katzan Jr.: Batch, Conversational and Incremental Compilers, Proc. SJCC. pp. 47～58, (1969).
35) H. Katzan Jr.: META PI-An Interactive On-line Compiler Compiler, Proc. SJCC. pp. 201～218, (1969).
36) R. Robert: HELP——A Question Answering System, Proc. FJCC. pp. 547～554, (1970).
37) J. Reinfelds et al.: AMTRAN——An Interactive Computing System, Proc. SJCC. p. 537, (1970).
38) H. A. Elder: On the Feasibility of Voice Input to an On-Line Computer Processing System, Comm. ACM, Vol. 13, No. 6, p. 339, (1970).
39) J. Reinfelds: AMTRAN 72, Proc. 1st USA-Japan Comp. Conf. p. 109, (1972).
40) T. Takeshita: Survey of Programming Languages in Japan, Proc. 1st USA-JAPAN Comp. Conf. p. 231, (1972).

(昭和47年11月16日受付)