Vol. 19 No. 100　　電波研究所季報　　January 1973
pp. 3—20

研　　究

UDC 681.3

入出タイプライターを用いたオンライン・ターミナルの設定とそれによるリアルタイム処理の実例

柴田　久*，有馬安春*，高橋寛子*，柿沼淑彦*

(昭和 47. 11. 22 受理)

DESIGN AND WORKS OF ONLINE TERMINAL WITH I-O TYPEWRITER

BY

Hisashi SHIBATA, Yasuharu ARIMA, Hiroko TAKAHASHI,
and Yoshihiko KAKINUMA

A simple online terminal with only one I-O typewriter was designed, and several works on it were established for practical use. Here are explained the construction of this terminal and details of these works.

This terminal is built upon one partition with memory 68 K (1K=1,024 chs.). Precisely, in our computer system there can be 3 partitions prepared, and one of these is used for this terminal. The two others are usually appropriated to the service of normal batch job and system input reader. These three partitions can execute their respective work in parallel at the same time.

The peripheral devices used for this terminal are only one I-O typewriter, one mass storage device, and one magnetic tape unit. When circumstances require, an optical paper tape reader and card punch unit are used sometimes additionally. The card reader is absolutely necessary as System Input Unit (SIU), but at this terminal it is only used for the input of Monitor Control Directories at the opening step.

Actual operation of this terminal is preceded by two ways, i.e., one is by typewriter input and the other is that program is loaded by monitor control directory. In the former way any program entried on user's file can be called as one likes. In the latter some system programs can be used as well as user's programs.

Actual works established at present are as follows:

(a) Mutual translation between Kana-moji and Rôma-ji,

(b) Detection of ionospheric hourly data,

(c) Program debagging

* 情報処理部　情報情報処理研究室

(1) Method by normal language processer,

(2) Method by special syntax checker,

(d) Separate execution of sections taken in a FORTRAN program,

(e) Interpretation and evaluation of mathematical formulae.

1 序

電子計算機のオペレーションには，周知のように，バッチ処理とリアル・タイム処理とがある。これらの処理方式にはそれぞれ特徴があり，取扱われる問題や作業の形体およびそれらを包む環境条件によってどちらがより有効かが決められる。

科学研究においては，一般にバッチ処理が向いていると言われている。理論から誘導された式の数値化，実験や観測から得られたデータの解析など，この分野では計算機は仕事の終盤に近い段階で事後処理のために使われることが多く，バッチ処理でもじゅうぶん間に合うわけであるし，また実際にその方がいろいろな面で取扱い易いことが多い。

しかし，科学研究の中にもリアル・タイム処理を絶対的に必要とし，もしくはそれが極めて有効と見られるような種類のものもある。また，普通の理論的研究や実験的研究においても，その研究のある過程に会話型リアル・タイム処理を取り入れることにより，研究方法は格段に改善され能率化を計り得ることが期待されるのである。

ただし，そのためには人と計算機との共同作業という認識のもとに，研究の進め方に対する綿密な計画を立てておくことが必要である。

リアル・タイム処理をオンラインで実行する方法としては，TSS (Time Sharing System) 方式がよく宣伝されており，通信回線からのキュー（呼）によって設定する方式も開発されている。それらの方式は実際に各方面で実用され，それぞれ効果を上げているのであるが，それを実現するためにはやはりかなりの設備と維持経費を必要とするばかりでなく，研究などにあってはむしろ利用技術の開発が遅れ方式面の先行が目立つこととなり，いきなりそれを本格的に取り入れることは危険であると言わなければならない。すなわち，研究所においても，研究改善の面や研究管理，業務管理などの目的で，将来必ずオンライン・リアル・タイム処理方式が取り入れられることになると思うが，それをどのような形のものにするかは利用面をよく考え周到な計画に基づいて決めるべきものである。

われわれは，このような観点からほとんど経費をかけずにリアル・タイム処理を実行する方式を開発し実現した。それはオンラインの入出力タイプライタ1台だけをその端末とするもので，指令やデータの入力および出力はこの1台のタイプライタによってなされ，計算機との対話もすべてそれを通じて行なわれることになっている。このような極めて簡単な機器構成のターミナルに，もち論多くを望むことは無理であろう。われわれとしては，ただその機能を最大限に発揮させ，曲がりなりにもリアル・タイム処理を可能として利用面の開発を計り，本格的な将来の構想への足がかりにしようとするものである。

リアル・タイム処理は，このように，各問題ごとにその処理の内容が検討され，その操作のプログラムが作られるべきものであるが，われわれとしてはそれらの個々の問題には立ち入らず，ユーザーが一般に使用して役に立つと思われる2，3の仕事を可能とするようにした。

以下，このターミナルの設定の方法とそれによるいくつかの仕事の内容とを説明する。説明文の中で現在当研究所で使用している計算機（NEAC シリーズ2200モデル500）と採用している OS（MOD 4）における用語が使われる部分もあるが，それは説明を明確にするため止むを得ないものである。そのような用語についての詳しい説明は紙面の都合上省略されたが，それに未知の読者にもある程度の理解が得られるよう配慮したつもりである。用語についての詳しい内容については参考文献(1)を参照されたい。

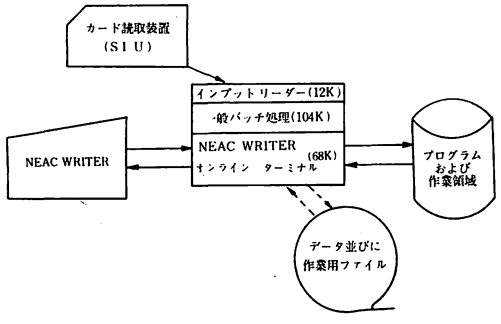
また，ここに説明される仕事は，すべて，当研究所の計算機システムという特殊条件の上に組み立てられたものであるが，他の一般の機種に対しても，そのままもしくは多少変更することにより，じゅうぶん転用することが可能であると考えられる。

2 設定方法と機器構成

このターミナルは，NEAC 2200 モデル 500 計算機の MOD 4 標準 OS の下で，一般のバッチ処理と並んで併行処理が出来るように設定される。すなわち，現在電波研究所の計算機の CPU（中央処理装置）メモリは256K（1K＝1,024字）であるが，モニター72Kを除いた残りの 184K が一般の仕事に利用出来ることになっている。このターミナルを開設するときには，この 184K は3つのパーティションに分割され，12Kをインプット・リーダ，104Kを一般バッチ処理，そして 68K がこのオンライン・ターミナルに振り向けられる。

このターミナルの設定はコントロール・カードを読み込むことによってなされる。すなわち，一般バッチ処理を実行させるためのコントロール・カードの間に，それらと同じ形式にそろえたこのターミナル用のカードをはさんで置き，カード読取装置により読み込ませるのである。そのとき，このターミナルのジョブは一般バッチ処理とは異なるパーティションのジョブ・クラスとして登録されるようにする。コントロール・カードが読まれると，それは JIF (JOB INPUT FILE) に入れられ，推定のパーティション上で実行されるようにスケジュールされてターミナルが設定される。このように，ターミナルのジョブとバッチ処理ジョブとは互いにパーティションが異なるため，両ジョブが同時に並行して実行されることが可能となる。もち論その場合，ＣＰＵ内の処理は両ジョブで時間を分けて行なわれることになるが，作業の性質上ＣＰＵ占有の優先度は普通オンライン・ターミナルの方に与えられる。

このターミナルの機器構成は第１図に示す通りである。この場合，入出力タイプライタとしてはオンラインの NEAC WRITER が使われ，それとターミナル用のプログラムおよび作業領域を入れた磁気ディスク装置１



第１図 機器の構成

台とで構成される。必要に応じて磁気テープ装置１台が割当てられることもある。もち論，もっと多くの磁気テープやライン・プリンターなどの周辺機器を割当てることも可能であるし，そうすることによりターミナルは非常に便利なものとなるのであるが，ここではバッチ処理との共存を考慮して，必要最小限の機器構成にした。これらの機器の他，カード穿孔装置やＰＴＲ（紙テープ読取装置）などバッチ処理であまり使用されない機器を使用することにはそれ程問題はないものと思う。

カード読取装置は，ターミナル用ジョブ設定の時，コントロール・カードその他のカードを読み込むだけのために使われる。それらのカードはインプット・リーダーを通じて磁気デイスクの JIF 領域内に入れられ，以後ジョブの実行につれてそこから順次引出されて処理されて行くので，ターミナル設定後はカード読取装置は完全に開放されることとなる。このインプット・リーダーの働らきは，一般のバッチ処理に対しても全く同じであって，ＪＩＦに入れられたいくつかのジョブは，モニターにより，パーティションごとにスケジュールされ順次に実行に移されることになる。

3 ターミナルの運用

このターミナルが設定された後は，いろいろなプログラムが呼び出され，次々と実行されることになる。このプログラムの呼び出し方法には２通りある。モニタ・コントロール・カードによる方法と NEAC WRITER からプログラム名を指示する方法とである。どちらの場合も実行されるべきプログラムは絶対番地形式で基礎プログラム・ファイルまたはユーザ・プログラム・ファイルに登録されているものでなければならない。この点，機に応じてターミナルからの指令により自由に仕事を動かし得る TSS や RJE (Remote Job Entry) などの本格的なオンライン・システムとは事情が異なるのである。

このターミナルの通常の運用は "PROG-CALL" というプログラム呼出用のプログラムを通じて，仕事に必要な処理プログラムを呼び出すことによって行なわれる。すなわち，PROG-CALL というプログラムが実行されると，次に実行させるべきプログラム名をタイプインすることが要求されるので，NEAC WRITER からそれを指定して必要なプログラムを呼び出して仕事を実行させるのである。この呼び出されるべきプログラムは，呼出用プログラム PROG-CALL と同じ MJB（プログラム格納領域の一つ）に入れられていなければならない。この MJB は，前節の機器構成の項で述べた専用の磁気デイスク内に設けられており，それに含まれる各プログラムはその処理終了後，自動的に再び PROG-CALL に戻るように作られている。したがって，そこでさらに別の処理プログラムを呼ぶことが出来る。いくつかのプログラムを実行した後，ターミナルの仕事を終えたい場合には，プログラム名の要求に対して空白を与えればコントロールはモニタに移り，JIF から次のコントロール・カードを読んでその指示にしたがうことになる。

コントロール・カードによる方法は，プログラムのロード・ユニットがいくつかに分散しており，フアイルのアサインを変更しながら仕事が行なわれるような場合に使われる。たとえば後に説明するプログラムのデバッギングの場合のように，モニタに含まれる標準の言語翻訳プログラムやリンケージローダを使う仕事では，この方法が取られることになる。この方法では，あらかじめ為すべき作業の順序を定めておき，１つ１つの作業をジョブ・ステップとしてコントロール・カードにより編成し

並べておく。順序に並べられたジョブ・ステップは，ある条件の設定によりスキップすることは出来るが，それもジョブ設定時の計画に基づくものであって，仕事の途中で任意の作業をはさむようなことは不可能である。しかし，それらのジョブ・ステップの1つとして，前段に述べた PROG-CALL を入れることは可能であるので，それとこのコントロール・カードによる方法とをうまく組合せれば，かなり融通性のある仕事を計画することも出来るわけである。

PROG-CALL による場合はもち論，コントロール・カードによる場合でも，このターミナルは設定時にカード読取装置を通じて制御用カードの読み込みが行なわれる他は，オペレーターの介入は全く必要が無く，ジョブ遂行のためのコントロールや作業に使うデータなどの情報はすべて JIF から読まれることになり，TSS や RJE 等のように自由且強力なものではないとしても一応オンライン的な効果は達成されることになっている。

4 仕事の実例

リアル・タイム処理のためのオンライン・ターミナルを研究活動に利用するためには，本来はそのテーマごとに問題に密着した特殊な方式が考案されるべきものと思う。しかし，それの中にもいろいろな問題の途中で共通的に使えて，準備しておけば便利と思われるような作業もある。われわれは，そのようないくつかの仕事を取り上げて，実用に供し得るプログラムとして完成した。すなわち

(a) カナ対ローマ字相互翻訳
(b) 電離層データの索引
(c) プログラムのデバッギング
　（i） ランゲージ・プロセッサー利用方式
　（ii） 会話型シンタクス・チェック方式
(d) 会話型ブロック別計算方式
(e) 数式計算

などである。以下，これらの仕事について項を分けて説明する。

4.1 カナ対ローマ字相互翻訳

電波研究所では，現在，一般の入出力にはカナ文字を扱っていない。それはハードウエア経費を節約する意味もあるが，使用ひん度の低いカナ文字を除いてラインプリンターの印字速度を上げるなど，科学技術計算を主体とする計算機利用の効率を高めるためでもある。

しかし，カナ文字は全く不必要というわけではなく，ハードウエアの都合上，止むなくローマ字で間に合わせている向きもある。NEAC WRITER の場合，カナ文字の入出力は可能であるので，ここではそれを通じてカナ文字が使える手段を与えることにしたのである。

実際には，この入出力はカナ対ローマ字の相互翻訳として行なわれる。すなわち，タイプインされたカナ文字は計算機内部メモリーや磁気テープや磁気ディスクなどのファイル内ではローマ字に翻訳されて保管されるし，NEAC WRITER に出されるときは，それがカナ文字に変えられてタイプアウトされる。このように計算機内部でローマ字として取扱われているため，カード・リーダーやラインプリンタなど他の周辺装置との関連も容易につけられることになるし，プログラム上での処理もすべてローマ字として扱えばよいわけである。

日本文をローマ字で綴るための文法はいろいろある。ヘボン式，日本式，訓令式などである。ここでは，それらのうち訓令式新表によることとした(附録1参照)。それは昭和29年内閣告示として公布されたもので，わが国の公式文書はすべてこれによることとなっており，学校教育でも主としてこれを教えることになっているからである。ローマ綴りとしては，訓令式などよりは古くからよく使われているヘボン式を可とする人も多く，それをこの文法の中に組み入れることも一応は考えたのであるが，英語などの外国語や国有名などをカナに変えないでそのまま取り出すような場合のために，あえてその組み入れは避けることにした。

訓令式新表には，第1表と第2表とあるが，ここでは第2表は採用せず第1表のみによることとした。また，NEAC WRITER の字種の都合から，"そえがき" の部分も止むを得ず一部変更されることになる。次にこの翻訳ルーチンを使用するための取り決めを列挙しておく。

(1) 日本語のローマ字綴りの文法は，訓令式新表の第1表による。ただし，その "そえがき" は適用せず，次の規定にしたがう。
　（i） はねる音「ン」はすべてNと書く。Nの次の母音またはYを切り離す必要がある場合には，Nの次に—（告示では'）を入れる。
　（ii） つまる音「ッ」は，次の音節の最初の子音字を重ねることによって表わす。
　(iii) 長音は母音字のあとに—をつけて表わす。
(2) 日本文は，カナ文字綴り，ローマ字綴りのいずれで書くにしても，必ず単語ごとに区切って書き，間に1個以上のスペースを置く。それは，翻訳の単位長を決めるためと読みやすさのためのもので，区切り方に特別の規定はない。
(3) ローマ字で綴られた単語が(1)の規定通りでない場合には，それは外国語または固有名詞とみなされ，カナ文字出力の指定があっても，翻訳されることなく，ローマ字綴りのまま出力される。

(4) つまる音の「ッ」や複合音シャ，シュ，ショなどの「ャ」，「ュ」，「ョ」をカナで表現する場合，一般には小文字が使われるが，NEAC WRITER にはそのような大小の区別は無いので，それらはすべて普通の文字と同じ大きさで出力される。カナ入力の場合には，これらを区別する必要があるが，そのためには半濁点をつけることにする。すなわち，「ッ」は「ツ」，「ャ」は「ヤ°」として入力される。

この相互翻訳のためには，2種類の変換表が必要である。その1つは NEAC WRITER のコードと計算機内部コードとの対応表であり，他の1つは内部コードのカナ対ローマ字の変換表である。

NEAC WRITER の字コードは，NEAC コードであり，内部コードとは別のものである。したがって，NEAC WRITER を通じての入出力においては，そのコードの対応表を置いて相互に変換してやらなければならない。このような関係は，カードやプリンタなどの場合でも全く同じであるが，それらの標準入出力装置に対してはモニタ（IOFCS）が面倒を見ているため，ユーザがそれを考える必要は全くないが，NEAC WRITER のような非標準機器の場合，先ず入出力指令そのものをコボルやフォートランで書くことは不可能であり，コード変換の作業も困難である。したがって，その入出力動作およびコード変換の部分はどうしてもアセンブラに頼らざるを得ない。実際に，この部分のプログラムは，入力と出力とに分けて，別々にアセンブラで書かれ，主としてコボルのプログラムから CALL することによって目的を達するようにした。

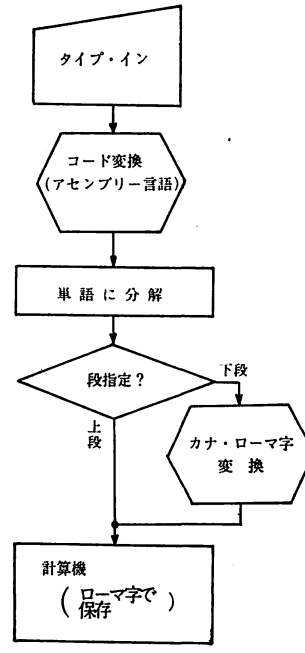
NEAC WRITER には，キー・ボードの上段と下段の指定があり，上段指定の場合は数字およびローマ字，下段指定の場合にはカナ文字がそれぞれ表わされることになっている。すなわち，文字コードは全く同じであっても，上段か下段かの指定により，外部に現われる文字は全く違うわけで，変換にはこの上下段の指定が重要な意味を持つことになる。

カナ文字とローマ字との相互翻訳プログラムはコボル語によって書かれており，そのため2つの表が作られた。カナ文字の表とローマ字の表である。カナ文字の表は，アイウエオ，カキクケコ，……，パピプペポの単音節のカナ5×15の表であり，ローマ字の方は母音だけの5文字と，△，K，S，……，Pの子音字15の2つの系列表の組み合わせによって表現する。このカナ文字とローマ字との対応は，カナの2次元配列表の（i，j）の要素と，ローマ字の母音列i，子音列の各要素の組み合わせとを当てることによってつけることにする。はねる音「ン」，つまる音「ッ」，複合音「キャ」，「ピュ」などについては，プログラム上の操作によって解決するようにした。

これらの翻訳操作は，ローマ字綴りを訓令式新表に限ったためその文字表現がきわめて規則的になっており，どちらの翻訳の場合も1文字づつ処理して行けばよく，明快かつ容易なものとなった。これら翻訳手続の流れ図を第2～5図に示しておく。

4.2 電離層データの索引

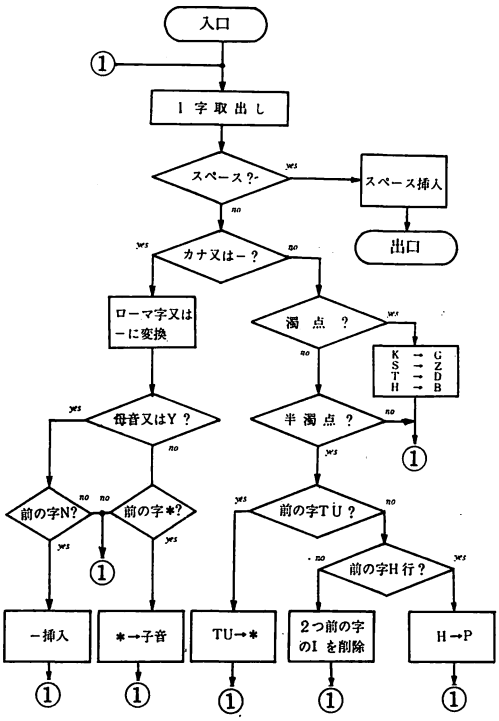
電波研究所においては，電離層を観測してデータを作り，国際的に交換するのも1つの重要な仕事となっているが，現在観測後読み取った値を国際交換用の月報の形式にまとめる作業は電子計算機によって行なわれている。



第2図

その作業では，ただ単に月報を作成するだけでなく，以後のこの分野の研究に役立たせるためのデータの蓄積が行なわれている。すなわち，電離層の各観測所，各特性ごとのデータが，1ケ月分ごとにまとめられて，磁気テープのファイルとして集積されることになっている。

磁気テープにファイルされた電離層データは，必要に応じて計算機に取り出されて使用される。それはバッチ処理でも可能であるが，その場合には選ばれるデータは当初の計画に基ずく固定されたものでなければならない。これに対して，リアル・タイム処理では，いくつかのデータを取り出して見たあとで，それらの値の出方に



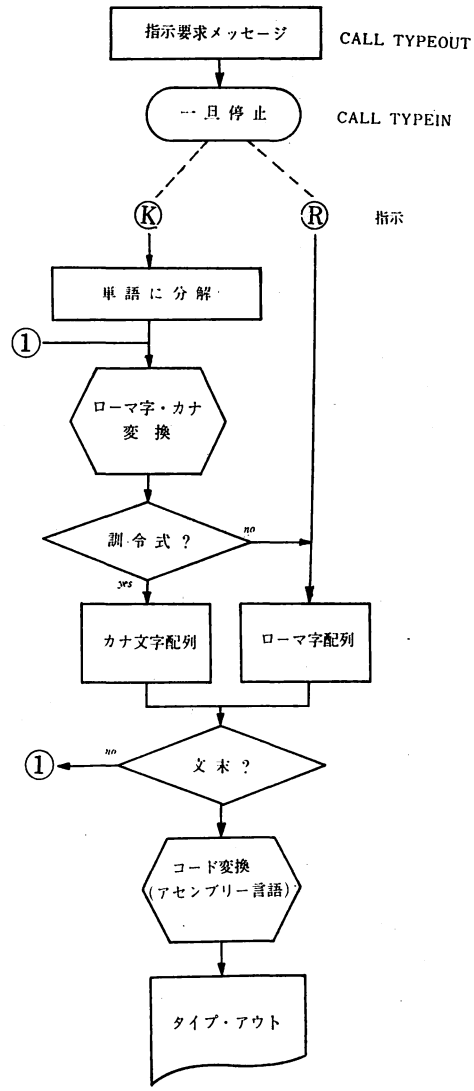
第3図

よって，次に取り出すべきデータが流動的に決められる利点がある。

実際のこの索引の仕事はきわめて簡単である。そのためには，毎月の月報作成ルーチンによって作られた電離層データ・ファイル(磁気テープ)を掛けて置き，NEAC WRITER から観測所，特性名，年月日，時刻を指示すればよい。観測所はその実際の名前（英字，2桁だけが有効）でも，数字コード（3桁）でも，どちらで指定してもよい。特性はコード（数字2桁）で指定する。年月日は西歴年（4桁），月および日（それぞれ2桁）をスペースを置いてこの順に数字で指定する。時刻は年月日の後にやはり数字2桁で指定するが，この指定は無い場合もある。すなわち，時刻が指定されておればその時刻に対応する時刻値のみが出されるが，時刻の指定が無ければ年月日で示された日の1日分のデータが出されることになっている。

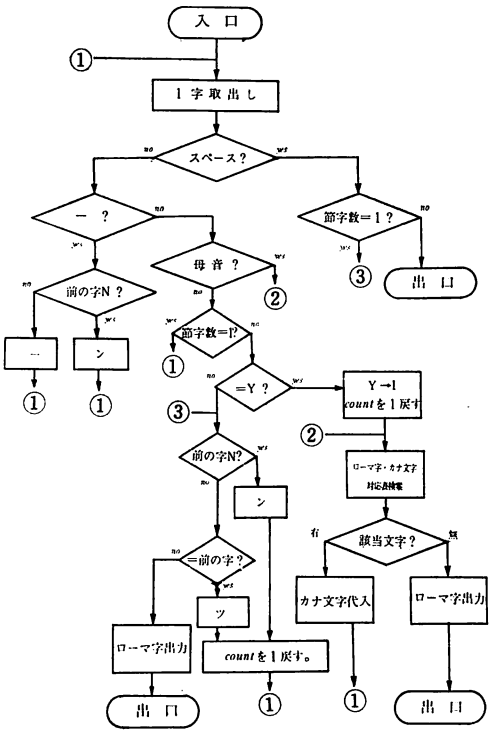
4.3 ランゲージ・プロセッサ利用によるプログラムのデバッギング

一般に，コーデイングを終った原始プログラムはデバッグの操作により点検修正される。その点検修正には2つの段階がある。すなわち，文法チェックと論理チェックとである。文法チェックは，規定形式のチェックであるので，適当なチェック用プログラムを用意することによって計算機にそれをやらせることが出来る。論理チェックは，文法チェックによって正しいと判断されたプログラムが果して計画通りの動作をするかどうかのチェックであり，これは実際にそのプログラムを動かして見なければわからない。



第4図

文法チェックや論理チェックで誤まりが発見された場合，その誤まりは修正されなければならないが，修正は機械に頼れるものではなくどうしても人がやらなければならない。ここで，人と計算機との対話が必要となる。



第5図

すなわち，計算機が点検の結果を人に伝え，人は誤まりを修正して再び計算機にチェックを依頼するのである。1つのプログラムを完成するまでには，このような点検と修正の対話は，普通，数回繰り返される。一般に行われているように，この操作をバッチ処理によることにすると，1つのプログラムのデバッグのために計算機室に数回足を運ばねばならない。

オンライン・リアルタイム端末が使える場合には，このデバッギングの操作は極めて簡単となる。すなわち，点検，修正，再点検の過程でかわされる計算機と人との対話を通じてリアルタイムで実行し，その場でプログラムを完成することが出来る。筆者らは，この点に着目し，リアルタイムによるプログラム・デバッギング方式を実現することに成功した。すなわち，前述（第2節）に述べたような簡単な機器構成のターミナルを通しての対話により，文法チェックと修正とを繰り返し，その場で正しい（文法的に）プログラムを完成させるのである。このようにして完成されたプログラムは，直ちにテスト・ランの実行に移すことも出来る。このテスト・ランの実行は，そのまま論理チェックにつながり，また答の見当をつける意味を持つことになる。

文法チェックは，前にも述べたように，計算機によってなされるのが普通である。そして，そのためにはプログラムをチェックするためのプログラムが必要である。この文法チェック・プログラムは，そのための専用のものとして作成されてもよいが，コンパイラなど翻訳プログラムにもその機能があるのでそれを利用する方法もある。専用のプログラムによるシンタクス・チェック方式については次節で説明することとし，本節ではモニタ内に用意されている標準の翻訳プログラムをそのまま利用したデバッギング方式について述べる。

ランゲージ・プロセッサを利用したプログラムのデバッギングは大別して3つの段階に分けられる。原始プログラムの読込，翻訳テストと修正および完成プログラムの作成の3段階である。

原始プログラムは，カード，紙テープ（NEACコードまたはTELEXコード）にパンチされたものが使われ，また NEAC WRITER 鍵盤からタイプインすることも出来る。プログラミング言語としては，現在のところ，FORTRAN, COBOL および ASSEMBLER の3種が使える。媒体がカードの場合には，プログラムのパンチ形式は各言語のものと全く同じである。紙テープの場合，カードと全く同じ形式にパンチされたものも取り扱えるようにはしてあるが，テープ長を縮めかつまた作成の便利さを考慮して，無駄なブランクを省いたパンチ形式のものでもよいようになつている。すなわち，紙テープや鍵盤からの入力の場合には，カード番号や命令文の前後のブランクを省略して，各言語ごとにそれぞれ簡略化形式が定められている。

プログラムの読込の段階では，カードや紙テープなどの原始ソース・プログラムが読込まれ変換され，磁気ディスクまたは磁気テープ内にカード・イメージのソース・プログラムが作られる。このカード・イメージのプログラムはこれ以後の各段階において，テストおよび修正が加えられ，完成プログラムの源となるものである。

翻訳テストおよび修正の段階では，翻訳テスト，SPR（System Printer Unit）テープの巻戻し，コンパイル・リストのプリント，誤まりの修正などの作業が行なわれる。翻訳テストは，前記のカード・イメージ・プログラム領域内のソース・プログラムを標準のランゲージ・プロセッサによって実際にコンパイルないしアセンブルして見るのである。この場合，翻訳されたプログラムが入れられる MGO（GO File）は不要であるので割り当てておく必要はない。ソース・プログラムのリストやエラー・メッセージなど翻訳プログラムからの情報は SPR に出されるが，この場合そのためにラインプリンタを使うわけにはいかないので，それには磁気テープが当てられる。プログラムに誤りがあるかどうかを知るために

は，この SPR の内容を NEAC WRITER に取り出す必要があるが，ユーザーのプログラムでこの SPR を直接操作することは出来ない。そこで，ここでは2重のアサインによってこの磁気テープを別の1つのファイルとして定義しておき，巻き戻しやメッセージの読取りなどの作業はコボル内でのファイルの取扱いとして実行することにした。メッセージを NEAC WRITER に取り出して誤りがわかった場合，その修正は鍵盤からのタイプインによって行なわれ，先の段階で作られたカード・イメージ・プログラムが修正される。

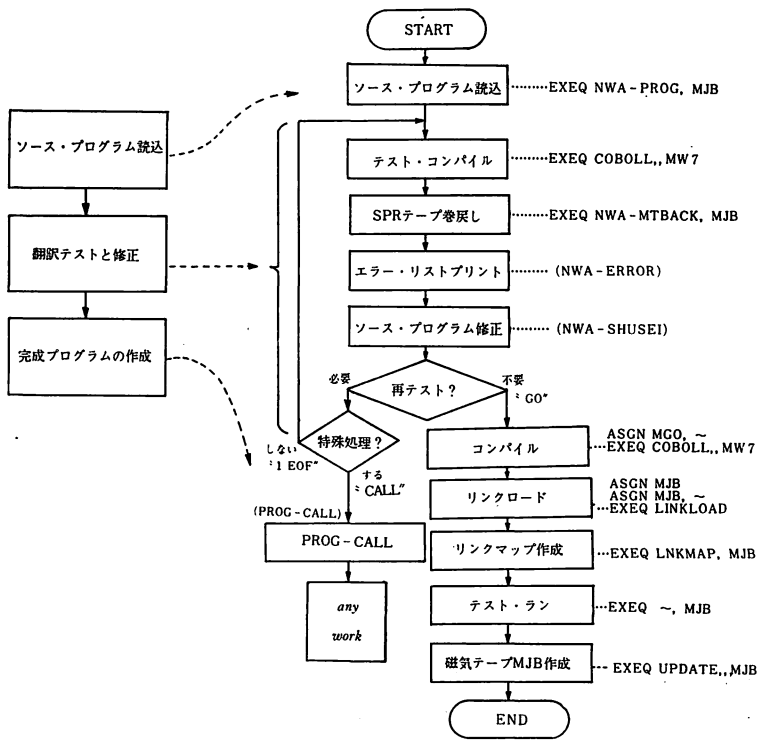
プログラムの修正が終った後，操作者は次の作業をどうするか選ぶことが出来る。すなわち，もう一度翻訳テストを行なうか，プログラムは完全であると見て完成プログラムの作成に移るか，あるいはここで特殊な処理を行なうか，のいずれかを指示するのである。その指示は NEAC WRITER 上で行なわれる。

再テストが指示されると，処理は翻訳プログラムに戻され，修正されたカード・イメージ・プログラムの翻訳テストが行なわれ，前と同様の動作が繰り返される。このターミナルの場合，この繰り返しは無制限に行なわれるわけではない。それはこのターミナルが本格的なオンラインを形成するものではなく，リモート・コントロールを自由に行なえるような形体にはなっていないためで，機器の制限や使用ＯＳなどの関係で止むを得ないことである。そこで，プログラム完成までにじゅうぶんと思われる翻訳テスト修正の繰り返し回数をあらかじめ見計らい，コントロール・カードを必要と思われる回数分だけ重ねておくことにした。すなわち，翻訳テストの繰り返しは，この重ねられたコントロール・カードの回数分まで行なわれる。この重ねられるコントロール・カードの枚数には別に制限はないので，実用的にはこれでじゅうぶんであると思われる。翻訳テストが数回で完了し，重ねられたコントロール・カードの途中で次の処理に移りたい場合には，残された翻訳テスト用コントロール・カードをすべてスキップするようにしておく。

翻訳テストによっては，プログラムの文法上の誤りテストが行なわれる。文法的には正しくても，それは必ずしも希望通りの仕事をするものとは限らない。すなわち，論理テストが必要である。そのためにはテストランを実行して見るのが最も手っ取り早いであろう。第6図にこのテストランまでを含めたこのデバッギング方式のフローチャートを示しておいた。

完成プログラムを作成し保存するためには，いろいろな方法が取られる。テストランまで実行したものであれば，直接実行プログラムをロード出来ようにしたＭＪＢ形式のオブジェクト・プログラムが作られているわけであるので，それを保存しておけば最も便利である。もち論，ＭＧＯ形式で保存することも可能で，そうしておけば他のプログラムとの連結も可能となる。ソース・プログラムが必要な場合には，カード・イメージで保管されているプログラムを実際のカードに打ち出すことも可能であり，そのためのプログラムもすでに用意されている。もっともそのようなカードのプログラム・デックは，また原始プログラムの差替えによっても簡単に得られるわけである。

特殊な処理の実行について，翻訳テストと修正の作業の後で NEAC WRITER からその旨の指示をすると，計算機は次にどのような処理をすべきかの指示を求めて来るので，操作者はそれをプログラム名で指示してやることにする。それはあらかじめこのターミナル用のプログラムとして作られ登録されているものでなければなら



第6図

ない。それには後述の会話型ブロック別計算のための処理などいろいろのものが予定されている。

4.4 会話型シンタクス・チェック方式によるプログラムのデバッギング

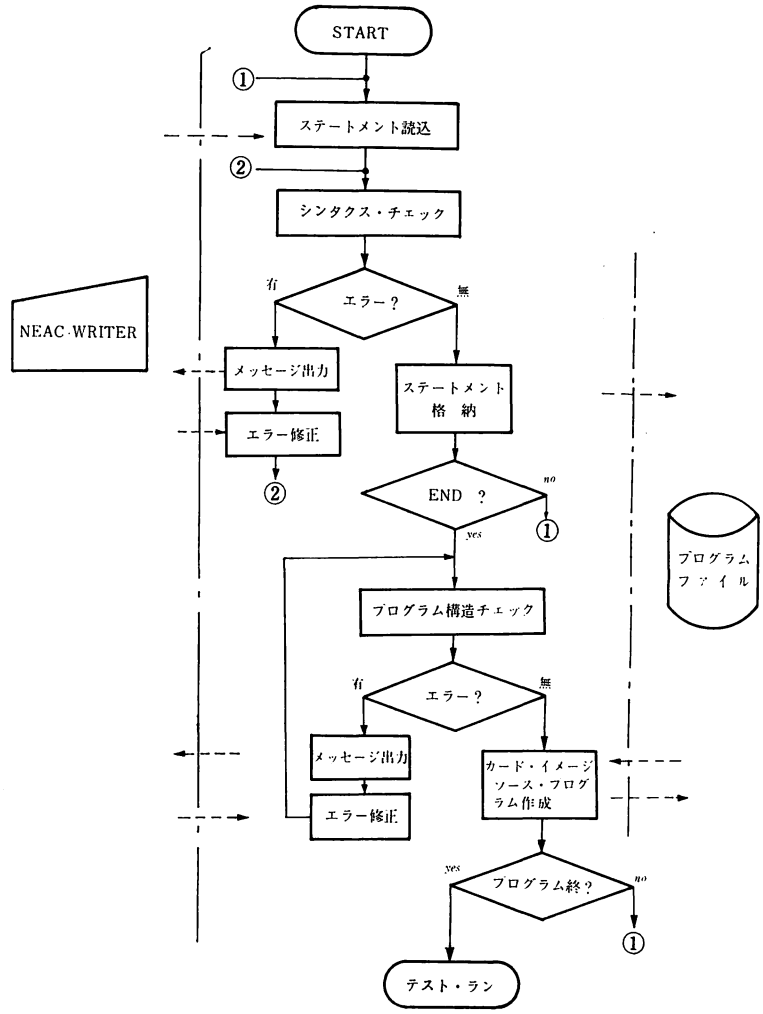
この方式は，前節にも述べたように，デバッギングのための専用のチェック・プログラムを作り，ユーザーのプログラムを1ステートメントごとに文法チェックし，計算機との対話により1つづつ修正しデバッグを完成して行く方法である。ステートメントごとの修正が完了すると，次には全体のプログラムに対して構造上のチェックがなされ修正も行なわれる。すべての誤りが修正されプログラムが完成すると，完成プログラムのリストが出力されデバッギングは終了する。ここで完成されたプログラムは，前節で述べたと全く同じ方法で，そのままテストランの実行に移すことも出来る。

現在のところ，この方式の適用出来るのはFORTRANのプログラムに限られる。すなわち，NEAC2200のFORTRAN L（JIS FORTRAN 7000の拡張）に準拠し，それに多少書き方の上で制限を附したものとなっている。

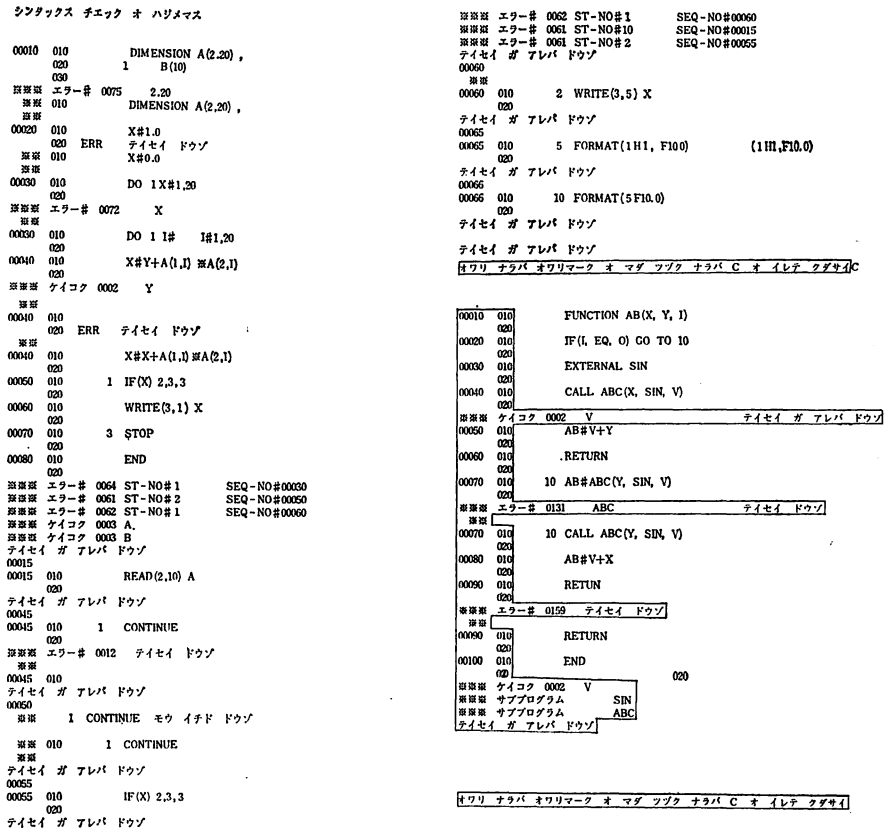
第7図にこの方式による処理の流れを示す。ステートメントはNEAC WRITERによりカードの形そのままでタイプインされる。読み込まれたステートメントは識別され，形式が合っているか，データ名，手続名等が正しく使われているか，型が正しいかなどのチェックがなされ，誤りがあればエラー・メッセージがNEAC WRITERに打出される。指摘された誤りは操作者により訂正され，再びNEAC WRITERからタイプインされることになり，シンタクス・チェックが繰り返され，誤りが無くなっ場合正しいステートメントが磁気デイスク内のファイルに入れられる。このファイルは，いったん入れられたステートメントがまた修正されたりあるいは間に別のステートメントがそう入されたりすることのため，ダイレクト・アクセスとされる。また，このシンタクス・チェックの段階では計算機内に変数や配列等のテーブル，ステートメント・ナンバーのテーブル，演算用のテーブルなどが作られ，このチェックに使われる他，次段の構造チェックにも利用される。

シンタクス・チェックによりプログラムの全部のステートメントがチェックされ，END ステートメントの格納が終ると，次にプログラム全体に対しての構造チェックがなされる。すなわち，前段で作られたテーブルを基として，ステートメント・ナンバーのチェック，DO文等ステートメント間のつながりに関するチェック，メイン又はサブプログラムの形式チェック等が行なわれる。チェックの結果誤りがあれば，メッセージが出され操作者の修正に応ずることになる。

シンタクス・チェックおよび構造チェックの詳しい内容は，それによって出されるエラー・メッセージの内容



第7図



第8図 会話型シンタクス・チェック方式によるプログラム・デバッギングの実行例

```
      DIMENSION   A(2,20) ,                    00010001
1         B(10)                                00010002
      READ(2,10) A                             00015001
      X#0.0                                    00020001
      DO 1  I#1,20                             00030001
      X#X+A (1,I) ※A(2,I)                      00040001
1     CONTINUE                                 00050001
      IF(X) 2,3,3                              00055001
2     WRITE(3,5) X                             00060001
5     FORMAT(1H1,F10.0)                        00065001
10    FORMAT(5F10.0)                           00066001
3     STOP                                     00070001
      END                                      00080001


      FUNCTION AB(X, Y, I)                     00010001
      IF(I .EQ.0) GO TO 10                     00020001
      EXTERNAL SIN                             00030001
      CALL ABC(X, SIN, V)                      00040001
      AB#V+Y                                   00050001
      RETURN                                   00060001
10    CALL ABC(Y, SIN, V)                      00070001
      AB#V+X                                   00080001
      RETURN                                   00090001
      END                                      00100001
```

第9図 会話型シンタクス・チェック方式により完成されたプログラム

につながるものであるが，エラーには番号がつけられており実際のメッセージはこの番号でタイプアウトされる。このエラー番号およびエラーの内容は表として附録に示される。

プログラムが完成されてＥＮＤ文が格納され終ると，ダイレクト・アクセス・ファイル内のソース・プログラムはカード・イメージの形で磁気デイスク内のシーケンシャル・ファイル内に移し替えられる。このカード・イメージ・プログラムはコンパイラーによって直接コンパイルすることが出来る。

１回のデバッギングでいくつかのプログラムを同時にテストし完成したいことがある。そこで，上のような手続により１つのプログラムが完成した時，後にまだプログラムが残っているかどうかがきかれ，残っている場合にはそのプログラムのデバッギングをすることになる。プログラムが全部終了した場合には，テストランのためのコンパイルなど，次の作業に移って行くことになる。

第８図にこの方式によるデバッギングの１例を示す。図中線で囲ってある部分が計算機からの出力であり，その外の文は NEAC WRITER からタイプインしたものである。デバッグが完了すると，カード・イメージ・ファイルが作られると同時に完成プログラムのリストが出力されるが，第９図がそのリストである。

```
          C$ $ P00
0001          X=0
0002          Y0=1
0003          H=0.2
0004          E=1.0
          C$ $ P01
0005          CALL SDEQ3(1, X, Y0, H, E, D, F, 0)
0006          WRITE(3,1) H, Y0
0007        1 FORMAT(1H, 2HH=, E20.9,5X, 3HY0=, E20.9)
0008          EXTERNAL F
          C$ $ P02
0009          STOP
          C$ $ W  D
          C$ $ R  H,Y0, X
0010          END


0001          FUNCTION F(X, Y)
0002          F=3*Y/(1.0+X)
0003          RETURN
0004          END
```

第10図

```
¥P00-01
H#          .200000000E+00        Y0E    #       .799601214E+01
D#    7.9996933
H#0.05
X#0                      E-01
Y0#1
¥P01
H#          .500000000E 01        Y01    #       .799997878E+01
D#    7.9999986
¥E¥
イレナオセ
¥E
```

第11図

4.5 会話型ブロック別計算方式

この方式は，FORTRAN のプログラムをいくつかのブロックに分けておき，実行時に NEAC WRITER からの指示により，１ないし数ブロックを分けて実行させるようにするもので，実行指示の間に変数値の入れ替えや変数の値のタイプアウトなどの要求も出来るようになっている。

第10図がそのためのソース・プログラムの一例である。図中プログラムのステートメントの間に書かれたＣ＄＄Ｐ…がこのプログラムのブロック別を示すものであって，この例では00，01および02の３つのブロックが存在している。Ｃ＄＄ＷおよびＣ＄＄Ｒという記述もあるが，これらは NEAC WRITER に対してマニュアルで出力および入力の要求がなされる変数を定義しておくためのものである。

このように書かれたソース・プログラムを特別な変換用のプログラムで処理した後にコンパイルすると，ブロック別計算のためのオブジェクト・プログラムが得られる。第11図は，第10図のソース・プログラムから作られたブロック別計算プログラムの実行例である。すなわち，オブジェクト・プログラムの実行が始まると，先ず，プログラムのどのブロックを実行するかの指示要求が出るので，ここでは¥Ｐ00—01と指定し00から01までのブロックの実行を指示する。01ブロックでは WRITE 命令があるが，その出力はこの場合 NEAC WRITER になされるようにになっており，ここではＨとYoの値がタイプされた。ここでＤ#をタイプインすると，Ｄは Ｃ＄＄Ｗで定義されているため，その内容の出力を要求出来ることになっており，実際にその値が出て来ている。次に，Ｃ＄＄Ｒで入力可能として定義されているＨ，ＸおよびYoの値を代入し，¥Ｐ01により01ブロックを再び実行させるとＨおよびYoがプリントされる。そこでＤ（正確値8.0）の値は満足出来る精度となったので，実行を終了させるため¥Ｅをタイプし処理を終結したのである。この例で，¥Ｅ¥はタイプの入れ間違いであるので，計算機は「イレナオセ」と指示して来ている。この例の場合，プログラムの実行を終結させるためには¥Ｐ02によって02ブロックの実行を指示してもよいわけである。

以上の例題の説明によって，ブロック別計算方式がどのようなものであるかはわかったと思うが，実際に実行すべきブロックを指示するには４つの形式がある。すなわち

(ｉ)　¥Paa

(ii)　¥Paa—bb

(iii)　¥Paa，k

(iv) ¥Paa—bb, k

の4通りである。(i)はプロック aa の実行を指示するものであり，(ii)は aa から bb までに至るプロック全部の実行を指示している。(iii)および(vi)は aa および aa から bb までのプロックをそれぞれk回繰り返し実行させるための指示である。

C$$RおよびC$$Wは，NEAC WRITER からの指示により数値を入出力出来る変数を定義するためのものであることは前に述べたが，ここで定義出来る変数は整数型，実数型または複素数型の変数であり，また配列であってもよい。配列の場合には，DIMENSION 文などで定義されたと同じ大きさの次元宣言子をつけておかなければならない。

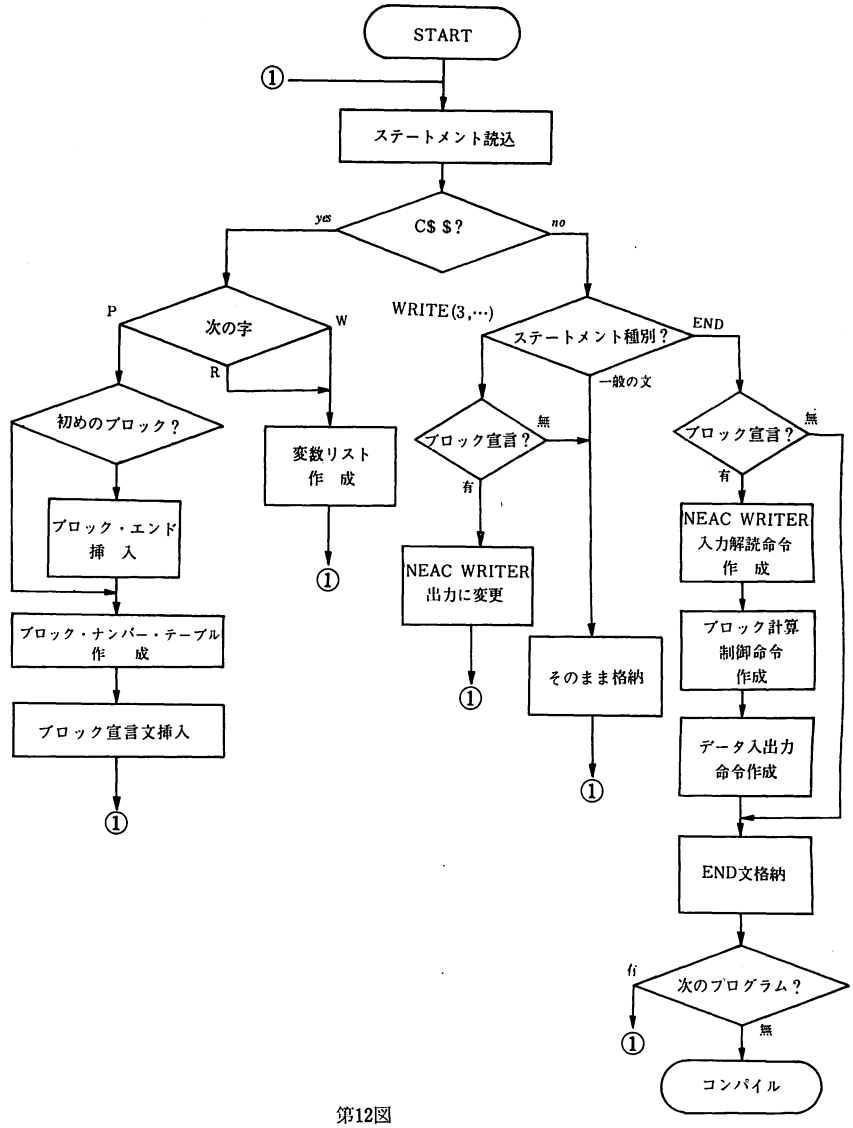
C$$PまたはC$$Wが読まれた場合には，NEAC WRITER からマニュアルで入力され，またはマニュアルでその値の出力が要求出来る変数別のリストが作成される。

WRITE (3, …) は，一般のプログラムではラインプリンタへの出力であるが，プロック別計算ではそれを NEAC WRITER に出すことにする。したがって，ブロックに含まれるこの命令は NEAC WRITER 出力の命令に変更される。

END 文は1つのプログラムの終了を示すものであるが，この文が読まれると変換処理プログラムはプロック別計算に必要な処理命令群をそう入する。それは，NEAC WRITER からの指示の解読，ブロック別処理実行の制御，入力または出力変数リストに載せられた変数への値の代入もしくは出力のための命令等である。これらの命令群のそう入が終る END と C$$P，C$$RおよびCSSWは，FORTRAN コンパイラではコメントと見られる。したがって，ソース・プログラムをそのままコンパイルすれば，それは普通の FORTRAN プログラムとして扱えるのである。

第12図に，このように書かれたソース・プログラムをプロック別計算プログラムに変換する処理手続の流れを示す。

読まれたステートメントがC$$Pを含む場合には，それはプロックの始まりを示すのであるので，そのための宣言文が作られ，同時にそのプロックのナンバー・テーブルが作成される。この場合，それがプログラムの最初のプロックであるときを除いて，前のプロックの終りを示す文が挿入せられる。



第12図

文が置かれ，そのプログラムは終了する。ここで次にプログラムがあれば再び次のプログラムの変換に戻り，それがないならば変換は完了し，オブジェクト実行のためのコンパイルに移ることになる。

これ以外の一般の命令文はこの変換によって何等影響を受けることなく，そのままソース・プログラムのステートメントとして格納される。

この変換を施すと，もとのソース・プログラムにはかなりの命令群が追加されることになり，変換されたプログラムは相当に大きなものとなる。実際に追加されるステートメントの数は次の計算式によって表わされる。

C$$R，C$$Wのリストに配列が無い場合：

$146+3n_W+4n_V+3n_P$

リストに配列が含まれる場合：

$276+3n_W+4n_V+3n_P+9n_{DW}+7n_{DR}$

ただし n_W ＳＰＲへの出力命令の数

n_V 入出力リストの変換の数

n_P Ｃ$$Ｐの数

n_{DW} Ｃ$$Ｗのリスト中配列の数

n_{DR} Ｃ$$Ｒのリスト中配列の数

これらを実行時の実際のメモリーに換算すると，各の値によっても多少変動はあるが，前者すなわち配列のない場合には約20K増，配列のある場合は28K増とみればよい。

4.6 数式の計算

算術式を NEAC WRITER からタイプインして，計算機にそれを解読計算させ，結果を NEAC WRITER に出力させるものである。

タイプ入力は，FORTRAN などと同様に，普通に使われている算術式そのままの形式でなされる。この式を構成する要素は第１表に示す通りのものであって，式中に第２表に示すような FORTRAN の基本組込関数が全部使われ，また別に自からが定義した関数を書くことも出来る。さらにまた必要ならば，あらかじめ作られているサブルーチン・プログラムをコボルのプログラムを通じて接続することも可能である。

この方式による数式計算実行の手順は第13図のように２つのステップがある。RUN ステップと DEF ステップである。RUN ステップは実際に計算を行なうためのものであり，DEF ステップは関数または定数を定義するもので，普通 RUN ステップの前に実行される。DEF ステップでは先ず DEF¥ という宣言を置き，たとえば

D(A, B, C)#B※B−4※A※C¥

PI#3.14159265¥

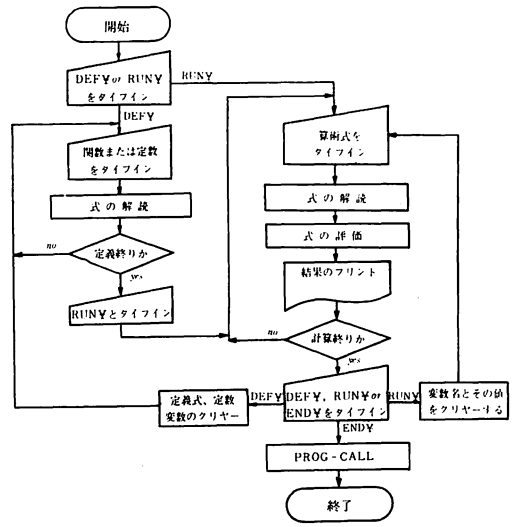
のように書けば，次の RUN ステップで関数 D と定数 PI が使えることになる。

第１表 数式を構成する要素

要 素	意 味
数 値	数字（０～９），符号（＋，－），小数点(．)，10のベキ（Ｅ）で構成
関数名	英文字で始まり，最大６文字までの英数字
定数名	英文字で始まり，２文字までの英数字
変数名	〃
#	左辺と右辺の連結記号，結果のプリント指示記号
(，)	関数とパラメータの区切り，または演算順序の指定
,	関数のパタメータとパラメータの区切り記号
¥	式と式の区切り記号
・～・	コメントの指示記号
	入力の終り符号
＋	加 算
－	減 算
*	乗 算
／	除 算
* *	ベキ算

第２表 関数一覧表

関数名	基本の関数	定 義
SQRT	平 方 根	$\sqrt{a}$
EXP	指 数	e^a
ALŌG	自然対数	$\log_e(a)$
ALŌG10	常用対数	$\log_{10}(a)$
ATAN	逆 正 接	$\tan^{-1}(a)$
SIN	三角法の正弦	sin(a)
CŌS	三角法の余弦	cos(a)
TANH	双曲正接	tanh(a)
AINT	切り捨て関数	aの最大整数≦|a|
AMŌD	剰 余	$a_1 \pmod{a_2}$
ABS	絶 対 値	|a|



第13図

RUNステップの宣言は RUN¥ であって，この後に

X#1.0/2.0※(=3.0+SQRT(D(1.0,3.0,2.0)))#

のような式がタイプインされると，この式の値が計算され，右側の#の次にその答がプリントされ，それと同時に変数Xにその値が格納される。この場合，算術式の右に#ではなく¥をタイプインすると，答の出力はされずにXへの代入だけが行なわれる。

変数値などの値は，計算機メモリ内では2進浮動小数点形式で保管されているが，これを出力する場合には10進数に変換され仮数部10桁のE型数として出力される。

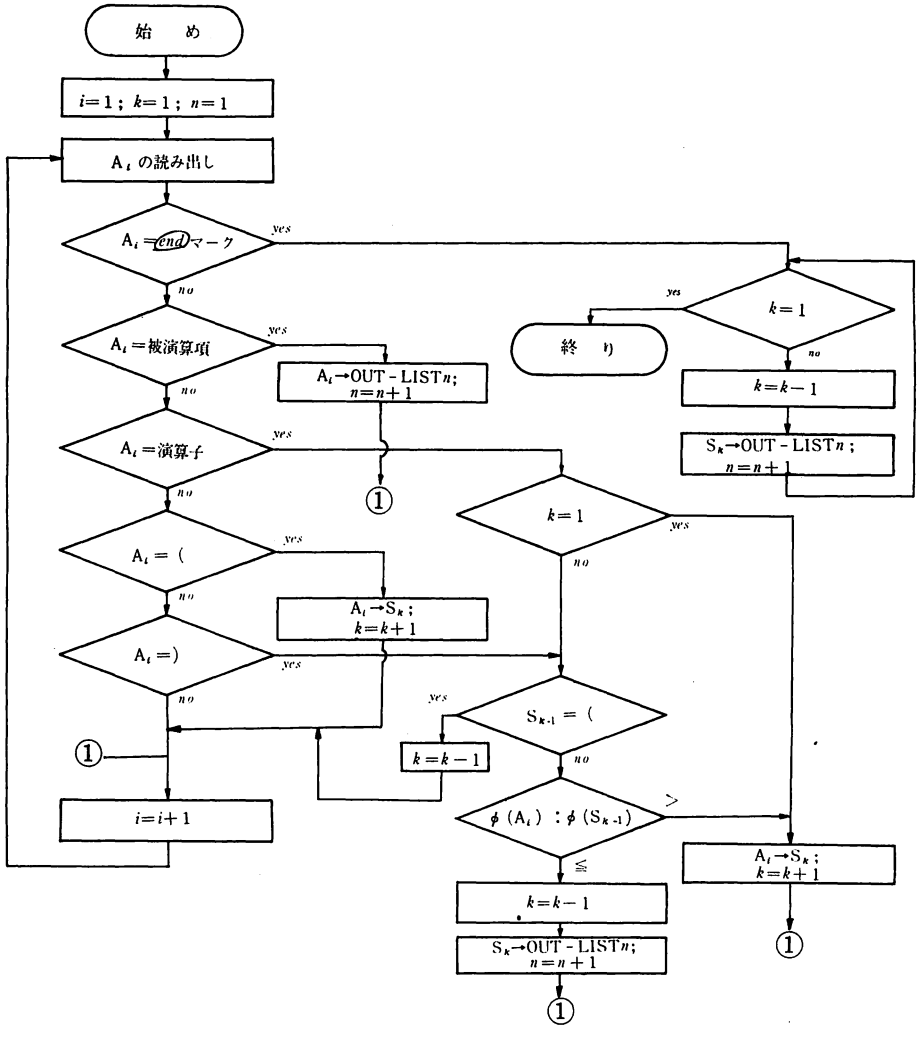
数式の解読および評価は入力された式を逆ポーランド記法に変換した上で行なうことにした。たとえば

a+b ⟶ ab+

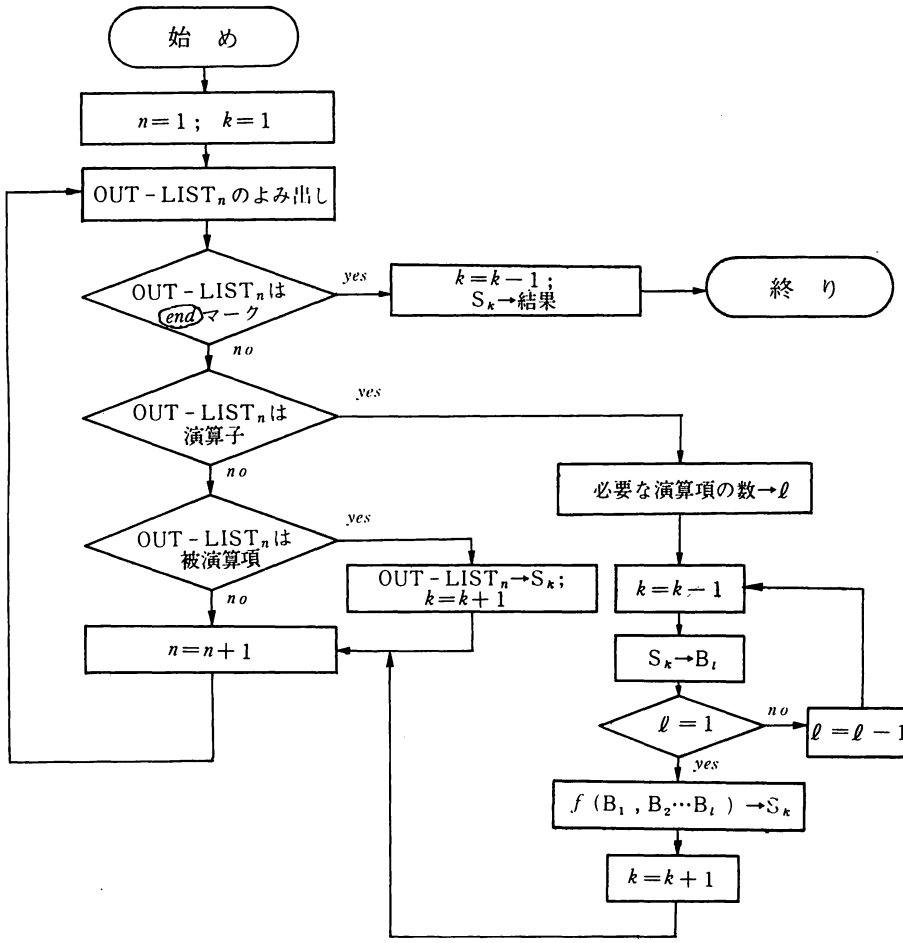
(a+b)*c ⟶ ab+c*

a+b/c ⟶ abc/+

のように変換するものである。この変換では被演算子の次に演算子が置かれるように変更され，それは演算する順番に並べられている。この変換のために演算子に優先順位をつけ，最も高いものは関数名であり，次に**，*または／，＋または－の順である。またメモリー内にスタックと称せられる領域を設け，演算子の優先性とスタックの性質とを利用して，逆ポーランド記法の中でのその順序とその位置が決められる。実際の変換の手順の流れは第14図に示される通りである。すなわち入力された数式の各項目 Ai は，被演算子であればそのまま出力リストに入れられ，演算子であればスタック（Sk）に入れられるが，その場合に演算子の優先度 ϕ の値が比較され，逆ポーランド記法上での位置が決められる。途中に



第14図



第15図

（が現われると，次の）が出るまでスタックの内容は凍結されることになる。この（はスタックの中にいったん入れられるが，凍結が解かれて演算子が出力リストに移される段階では記号列には残されない。

第15図はこのように逆ポーランド記法によって表わされ記号列を評価するための手続の続れを示すものである。スタック内には被演算子を演算子が出現するまで格納し，演算子が現われると計算が実行される。すなわち，その演算子が必要とする個数の被演算子（変数値）がスタックから取り出され，計算されその結果がまたスタックに戻される。そして次の記号へと読み進み，先の手順を記号列の全部について実行する。最終の結果はスタックの一番初め（S_1）の領域に出て来る。

この方式による演算に，多少複雑な処理を含むことが要求される場合には，別に作ったサブルーチン・プログラムを利用することになるが，そのためにはCOBOLのプログラムを仲介として接続することが可能である。この場合，この数式計算処理用のプログラムと仲介用のCOBOL プログラムとの間のデータ交換は IPC 領域（Inter Program Common Area）によって行なわれ，COBOL と FORTRAN の間の交換は磁気デイスクなどのファイルを通じて行なわれるようになっている。

5. 結 び

以上，NEAC WRITER だけを端末に持ち，極めて小規模な機器構成によって為し得るいくつかの仕事の内容について説明してきたが，前にも述べたように，それは本格的なオンライン・ターミナルではなく，いろいろな欠点を持っている。それは，本来バッチ処理用としてデザインされている標準ＯＳの下で行なわれるものであり，一般のバッチ処理と並行処理させるため極度に機器の制限をしていることから止むを得ないことである。その欠点のうち最大のものは，何といっても処理速度の遅いことである。erapsed time の大部分は，NEAC WRITER からの入出力の時間であり，それは現在の制限下では致し方がない。

次には専有メモリの問題である。現在の当研究所の内

部メモリは256K，そのうちモニタに72K，インプット・リーダに12Kが割当てられており，残りがこのNEAC WRITER オンラインと一般バッチ処理に向けられることになる。現在，前者に68K，後者に104Kと配分されてはいるものの，この容量はもち論両者共かなり窮屈な数字であると言わなければならない。

その他にも，調べればいろいろな欠点が浮んで来る方式ではあるが，兎に角曲がりなりにもオンラインでリアルタイム処理をここに可能としたわけで，目下さらにこれによって実行出来る仕事の種類を増すようなプログラムの開発に努力を傾けているわけであって，それらの仕事を通じて今後研究所におけるリアルタイム処理のあり方に対する指針ともなれば幸である。

最後に，この方式開発に当って深く理解され日頃指導鞭撻を戴いた尾方情報処理部長，ならびに計算機運用上いろいろ御迷惑を掛けながらも進んで御協力御尽力を賜わった計算機オペレータ諸氏に深甚なる謝辞を呈する次第である。

参考文献

(1) NEAC シリーズ2200 MOD 4 システム・モニタ説明書 其の他日本電気㈱ 計算機マニュアル

(2) 柴田久ほか；"電離層データの電子計算機による処理"，電波研季報，**14**，No.75，pp.594—608，Nov.1968.

(3) 柴田 久；"電離層データの電子計算による処理(続)"，電波研季報，**16**，No.82，pp.55—62，Jan. 1970.

附 録1

訓令式新表

第1表—（ ）は○出を示す—

a	i	u	e	o			
ア	イ	ウ	エ	オ			
ka	ki	ku	ke	ko	kya	kyu	kyo
カ	キ	ク	ケ	コ	キャ	キュ	キョ
sa	si	su	se	so	sya	syu	syo
サ	シ	ス	セ	ソ	シャ	シュ	ショ
ta	ti	tu	te	to	tya	tyu	tyo
タ	チ	ツ	テ	ト	チャ	チュ	チョ
na	ni	nu	ne	no	nya	nyu	nyo
ナ	ニ	ヌ	ネ	ノ	ニャ	ニュ	ニョ
ha	hi	hu	he	ho	hya	hyu	hyo
ハ	ヒ	フ	ヘ	ホ	ヒャ	ヒュ	ヒョ
ma	mi	mu	me	mo	mya	myu	myo
マ	ミ	ム	メ	モ	ミャ	ミュ	ミョ
ya	(i)	yu	(e)	yo			
ヤ	(イ)	ユ	(エ)	ヨ			
ra	ri	ru	re	ro	rya	ryu	ryo
ラ	リ	ル	レ	ロ	リャ	リュ	リョ
wa	(i)	(u)	(e)	(o)			
ワ	(イ)	(ウ)	(エ)	(オ)			
ga	gi	gu	ge	go	gya	gyu	gyo
ガ	ギ	グ	ゲ	ゴ	ギャ	ギュ	ギョ
za	zi	zu	ze	zo	zya	zyu	zyo
ザ	ジ	ズ	ゼ	ゾ	ジャ	ジュ	ジョ
da	(zi)	(zu)	de	do	(zya)	(zyu)	(zyo)
ダ	(ジ)	(ズ)	デ	ド	(ジャ)	(ジュ)	(ジョ)
ba	bi	bu	be	bo	bya	byu	byo
バ	ビ	ブ	ベ	ボ	ビャ	ビュ	ビョ
pa	pi	pu	pe	po	pya	pyu	pyo
パ	ピ	プ	ペ	ポ	ピャ	ピュ	ピョ

第2表

sha	shi	shu	sho	（標準式）
シャ	シ	シュ	ショ	
		tsu		
		ツ		
cha	chi	chu	cho	
チャ	チ	チュ	チョ	
		fu		
		フ		
ja	ji	ju	jo	
ジャ	ジ	ジュ	ジョ	
di	du	dya	dyu	dyo （日本式）
ヂ	ヅ	ヂャ	ヂュ	ヂョ
kwa				
クヮ				
gwa				
グヮ				
				wo
				ヲ

内閣告示第一号

国語を書き表わす場合に用いるローマ字のつづり方を次のように定める。

昭和二十九年十二月九日

内閣総理大臣 吉田茂

1 一般に国語を書き表わす場合は，第1表に掲げたつづり方によるものとする。

2 国際的関係その他従来の慣例をにわかに改めがたい事情にある場合に限り，第2表に掲げたつづり字によってもさしつかえない。

3 前二項のいずれの場合においても，おおむねそえがきを適用する。

そえがき

前表に定めたもののほか，おおむね次の各項による。

1 はねる音「ン」はすべてnと書く。

2 はねる音を表わすnと，次にくる母音字またはyとを切り離す必要がある場合には，nの次に'を入れる。

3 つまる音は次の音節の最初の子音字を重ねて表わす。

4 長音は母音字の上に'をつけて表わす。なお，大文字の場合は母音字を並べてもよい。

5 特殊音の書き表わし方は自由とする。

6 文の書きはじめ，および固有名詞は語頭を大文字で書く。なお，固有名詞以外の名詞の語頭を大文字で書いてもよい。

附録2 エラーナンバーの内容一覧表

エラーナンバー	内容
0001	ステートメントが20行以上よりなっている
0002	コメントが2行以上にわたっている
0003	ステートメントの1行目に継続印がある
0004	継続行に継続印がない
0005	数式の左辺に定数を使用している
0006	数式の左辺に使用不能の記号を使用している
0007	配列，関数名等を変数として使用している
0008	関数名の二重定義
0009	数式関数のパラメータに数を使用している
0010	数式関数の左辺に使用不能の記号を使用
0011	数式の左辺に余分の情報がある
0012	ステートメントナンバーの二重定義
0013	配列名を使用できないところに使用している
0014	関数，サブルチン名を使用できない
0015	数式中の括この数があわない
0016	使用不能の文字を含んでいる
0017	6文字以上のデータ名
0018	・の使い方に誤りがある
0019	論理演算の operater に誤り
0020	括この使い方に誤りがある
0021	数式関数の実際のパラメータに配列名等使用
0022	数式関数のパラメータの数が合わない
0023	, の使い方に誤りがある
0026	非論理型に論理演算を行なっている
0027	NOT の使い方に誤りがある
0029	.EQ., .NE. 以外の関係演算に複素数型を使用
0031	非数値型データに対して算術演算を行なっている
0033	巾に複素型，論理型，文字型を使用
0034	底に論理型，文字型を使用
0035	底が複素数の時巾は整数でなければならない
0036	添字に使われている定数に文法違反
0037	添字に使われているデータ名に文法違反
0038	添字に整数型でないものを使用
0039	添字に配列，関数名等を使用している
0040	添字満算で±以外の演算子を使用している
0041	添字演算で±の後は，定数でなければならない
0042	添字演算で＊以外の演算子を使用している
0043	添字演算で＊の後は変数でなければならない
0045	配列の次元が定義してあるものより大
0048	文字定数に文法違反がある
0049	指数部に使用不能の記号がある
0050	指数に数字以外のものを含んでいる
0051	数値型定数に不正の文字を含んでいる
0052	演算要素がぬけている
0053	演算子がぬけている
0054	実引数にEXTERNALでない関数名を使用している
0055	度数を関数のように使っている
0055	頭文字が英字でないデータ名を使用している
0057	データ名に英数字でないものを含んでいる
0059	サブプログラムに RETURN 文がない
0061	指定されたステートメントナンバの文がない
0062	入出力文が FORMAT 文以外を参照している
0063	入出力以外の文が FORMAT 文を参照している
0064	DO の端末文に使用できない文を使っている
0065	DO の端末文が DO 文より前にある
0068	DO の重なりに誤りがある
0069	STOP 文の STOP の後の情報に誤りがある
0070	文の番号の参照に文法違反がある
0071	計算型 GO TO 文に文法違反がある
0072	整数型変数でないものを不正に使用している
0073	変数名を配列として定義している
0074	次の数が3以上の配列を定義している
0075	配列宣言で次元の大きさの書き方に文法違反
0076	整合寸法の配列宣言に文法違反がある
0079	配列文の並びの書き方に誤りがある
0080	情報が欠けている
0081	データ名がぬけている
0082	IF ステートメントに文法違反がある
0083	数式 IF 文で複素型，非数値型データを使用
0084	論理 IF 文の後に使用できない文を書いている
0085	ステートメントナンバーの書き方に文法違反
0086	主プログラムに RETURN 文がある
0087	FORMAT 指定を必要とするのにぬけている
0088	関係演算に論理型を使用
0089	関係演算で文字型を他の型と比較している
0090	ステートメントナンバに数字以外を使用
0091	数式関数にステートメントナンバーがある
0092	論理型，文字型以外のものを論理型に代入
0093	文字型以外のものを文字型に代入している
0094	機種指定，FORMAT 指定に文法違反がある
0095	機種指定に使えない定数を使用
0096	入出力文の書き方に文法違反
0097	機種指定を入力文に3，5出力文に2としている
0098	入力文で END ERR の書き違い
0099	入力文で END ERR の後の文番号参照に文法違反
0100	入出力文で FORMAT 指定に配列でないデータ名使用
0101	入出力文のリストに定数を使用
0102	入出力文のリストに使用不能の記号使用
0103	入出力文のリストに関数名を使用
0104	入出力文のリストの区切り方に誤りがある
0105	入出力文のリスト中の括この使い方に誤り
0106	入力出文の DO 型ならびに文法違反がある
0109	論理型を非論理型に代入している
0110	複素数の書き方に誤りがある
0111	関数名がぬけている
0112	変換要素 E, F, G, D に於てwとdの関係に誤り
0113	変換要素 E, F, D に於てw. dの形でないもの使用
0114	FORMAT 中に使用できない符号を使用

0115	FORMAT 文に文法違反がある
0116	変換要素間の区切りに誤りがある
0117	FORMAT 中の括この数が合わない
0118	FORMAT 文にステートメントナンバーがない
0119	数値定数がぬけている
0120	FORMAT 文中に 3 重以上の括こがある
0121	そう入ステートメントの数が多すぎる
0123	入出力文のリストに未定義の配列名がある
0125	入出力文に機種指定がない
0126	出力文に ERR END の項がある
0127	DO 文の初期値，終値，きざみ値が整数型でない
0128	配列名を EXTERNAL として宣言している
0129	変数名を EXTERNAL として宣言している
0130	サブルーチン名でないサブプログラムを CALL
0131	サブルーチン名を関数名のように使用
0132	型宣言文の書き方に文法違反がある
0133	IMPLICIT 文のリストに文法違反がある
0134	IMPLICIT 文のリストの区切りに誤りがある
0136	文字型を倍精度または複素型に代入
0137	RETURN の後に余分の情報がある
0138	関数サブプログラムで関数の値が未定義
0139	型宣言に矛盾がある
0140	IMPLICIT 型宣言に矛盾がある
0141	コモンのラベルに文法違反がある
0143	COMMON 文のリストに関数名 サブルーチン 名がある
0144	COMMON 文に同一名が二重に 又は 仮引数がある
0145	文字定数で：がペアになっていない
0146	COMMON 文で宣言されている配列 が 整合寸法をもつ
0147	配列の二重定義
0148	コモン文の書き方に文法違反
0149	サブプログラムの仮引数の書き方に文法違反
0150	SUBROUTINE 文 FUNCTION 文がプログラムの始めにない
0151	：の使い方に文法違反
0152	数式関数名に変数名を使用
0153	CALL 文の書き方に文法違反
0154	数式関数を EXTERNAL と宣言している
0155	サブプログラムの仮引数に括こがある
0156	サブプログラムの書き方に文法違反
0158	FUNCTION 文に仮引数がない
0159	解読不能のステートメント

警 告	内 容
0001	FORMAT 文以外の非実行文のステートメントナンバー無視
0002	未定義の変数を使用
0003	配列の定義をしてあるが値が未定義