中型機による FORTRAN IV プログラムのエラーチェック*

後藤以紀** 大竹政光** 石綱豊子** 井上郁代**

1. 序　　言

1.1 意　　義

東大大型センターに設置されている大型計算機を用いれば僅か数秒で結果が得られるようなプログラムでも，思いがけない文法の誤りやせん孔の誤りなどにより，なかなか一度では通らないものである．大型センターに送って，はじめてそのような誤りを検出するようでは，利用者にとってその時間的損失は甚だしい．それを防ぐには，利用者側でのできうる限りのプログラムのチェックが必要である．さらにそれを手元にある既設の中小型計算機で行なえれば非常に便利である．このような立場から，明治大学に設置されている OKITAC 5090 M 中型計算機を用いて，HARP 言語で書かれたプログラムのチェックプログラムの開発を行なった．本論文はそのチェックプログラムについて述べている．

なお，この研究は，文部省科学研究費総合研究「電子計算機の全国的総合使用法の研究」の一環として行なわれたものである．

1.2 概　　説

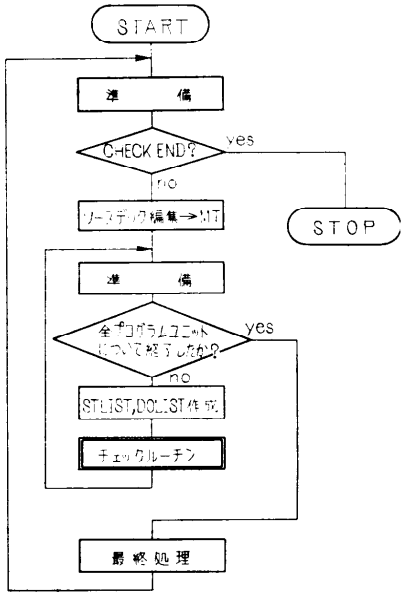
使用機器

（1） OKITAC 5090 M
（2） カード読み取り装置（CR）　1台
（3） ラインプリンター（LP）　1台
（4） 磁気テープ装置（MT）　2台
（5） 電動タイプライター　1台

この計算機のメモリーは8,000語（1語51ビット，含サイン，フラグ，パリティビット）まで可能であるが，明大に設置されているものは4,000語しかなく，そのために十分なチェックは難しい．

処理手順略解

さて，HARP 言語で書かれたソースデックをチェックする際に，$ カードやバイナリデック，あるいはデータなどを逐一除いていたのでは面倒でもあるし，まぎれて紛失しやすく，不備な状態でデックが大型センターに送られる恐れもある．そこで次のようにして大型センターに送るままの形でチェックを行なうことにした．



第1図　全体の流れ図

一般に一つの仕事を実行するプログラムデックは，複数個のプログラムユニットから成っており，それを1ジョブ（データデックがあればそれも含む）と呼ぶことにして，各ジョブを区別するために73カラムから JOBEND とせん孔したコントロールカードを用いる．さらに一番最後に計算機を止めるために，やはり73カラムから CHECKEND とせん孔したコントロールカードを用いる．このようなデックがセットされると第1図に示すような手順で処理されてゆく．

はじめにチェックを開始する種々の準備を行ない，次に全ジョブについてチェックを終了したかどうかを示すコントロールカードの判定をする．ソースデックの編集部ではソースデックのイメージを一連番号とと

* Error Check of FORTRAN IV Programme by Medium-scale Computer, by Motinori Goto, Masamitsu Ohtake, Toyoko Ishizuna and Ikuyo Inoue (Meiji University)

** 明治大学計算センター

もにLPに出力し，ソースデックを編集して1ジョブ分MTに書き込む作業，それに付随してCALLステートメント，FORMATステートメントの仮分類を行なっている．それが終了すると1ジョブのチェックを行なう準備をし，未チェックのプログラムユニットが残っているかどうか調べ，残っていなければ1ジョブのチェック終了処理（主としてエラーリストの出力）をしてはじめに戻る．

チェックは1プログラムユニットごとに行なうので，それに先だってそのプログラムユニット内で用いられているステートメントナンバーおよびDOステートメントの参照表を作成する．チェックルーチンではステートメントの分類およびそれに従って各ステートメントの処理が行なわれるが，HARP 5020では許されているが大型センターでは使用できないステートメントがあること，同一機能をもった数種のステートメントの混用はなるべく避け，標準的なもの1種を用いるのが望ましいこと，特殊な場合以外はMTや，DRUMに関するステートメントは用いない方がよいことから，使用禁止のステートメント，やむを得ぬ限り使用を避けるのが望ましいステートメントを検出している．使用を避けたいステートメントとして下記のものを検出する．

OVERFL (j), DVCHK (j), IF ACCUMULATOR OVERFLOW n_1, n_2, IF QUOTIENT OVERFLOW n_1, n_2, IF DIVIDE CHECK n_1, n_2, STOP n, DRUMに関するすべてのステートメント（ただし，チェックの都合上READ DRUMは使用禁止のステートメントとして検出されてしまう），READ INPUT TAPE k, n, リスト(同前)，WRITE OUTPUT TAPE k, n, リスト, BACK SPACE k, REWIND k, ENDFILE k.

使用禁止のステートメントについてはエラーナンバー表参照．

見い出されたエラーの位置はプログラムユニットのデック名とステートメントの左端に打たれた一連番号により，またエラーの種類は4桁のエラーナンバーで知れる．それは下記のように第1桁目で大分類されている．

0 コントロールカード（$カード）
1000 算術ステートメント
2000 論理ステートメント
3000 制御ステートメント
4000 入出力ステートメント
5000 仕様ステートメント
6000 サブプログラム，函数ステートメント
7000 その他
8000 用いない方が望ましいステートメントか，誤せん孔のステートメント
9000 使用禁止のステートメント

2. 各部分の説明

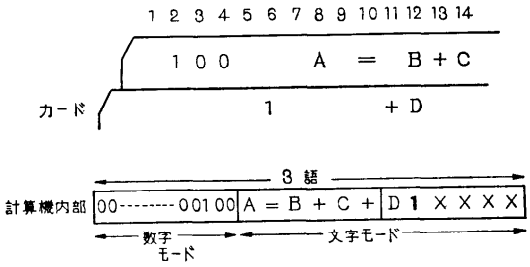
具体的にどのように処理しているか，要点のみを本節で述べる．

2.1 ソースデックの編集

この部分は，一つのステートメントの空白を詰め，MTに格納するルーチン，カードイメージをLPにより出力するルーチン，コントロールカード（$カード）処理ルーチン，コンティニュエーション処理ルーチンを支配下に持っている．まず，チェックを始める準備（初期設定）をし，CRによりカード1枚を読み込んで，計算機を止めるのか，チェックルーチンへ制御を移すのか判定する．コメントカードはLPに出力するだけで読み捨てられるが，$カード，コンティニュエーションの行はそれぞれ処理を受ける．これらのどれでもなければ一つのステートメントのイニシャルラインと見なされ，LPに出力され空白を詰めるルーチンへ飛び，MTに書き込まれ，再び前のカード読み取り部分に戻される．$カードであればその種類に従って，$DK-ENDが来るまで読み捨て（$LOADER），データデックが尽きるまで読み込まれLPに出力し（$DATA）デック名があるかどうかチェックし（$ HARP, $ ENTRY），このカードにせん孔されたデック名をメインプログラムとして登録（$ ENTRY）する．第1カラムに$がせん孔されていて以上のどれでもなかったら誤りと考えて，エラーリストに登録する．

コンティニュエーションの処理では第6カラムが数字かどうか，直前のカードの第6カラムを調べて数上りになっているかどうか，すでに10行を越えているかどうかなどを調べている．イニシャルラインはすでにMTに格納されているので，それを一度読み出してコンティニュエーションの行をLPに出力しながら全部つなぎ，再び空白を詰めるルーチンへ飛びMTに格納される．これでコンティニュエーションの行を含んだ一つのステートメントの編集が終る．ただし，FORMATステートメント，CALLステートメントはカードイメージどおりにMTに格納される．この

理由はこれらのステートメント中に現われた空白が意味をもつ場合があるからである．編集された例を**第2図**に示す．



第2図　編集された一例

1はステートメントの終了の目印．×は以前の状態により一般に何が入っているかわからない．$カード以外第1語目にはステートメントナンバーが入る．それがなければ0が12桁入る．1語中に文字モードなら6文字，数字モードなら12桁格納できる．

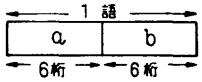
LP 出力において左端に印刷される一連番号は，$ LOADER, $DATA 以外の$カードおよび各ステートメントのイニシャルラインかデータデックに限る．$ LOADER のデックはそのようなデック名のバイナリデックが存在することを知らせるのみである．

2.2 空白を詰め MT に格納

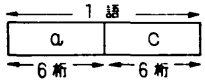
FORMAT ステートメントや CALL ステートメントのコンティニュエーションラインが，このルーチンに入って来た時には意味のある空白も詰める恐れがあるので，すぐ MT 格納へ飛ばしてしまう．それ以外のコンティニュエーションラインは 1～5 カラムのチェックを受けて空白を詰める処理をされて MT に格納される．ステートメントのコンティニュエーションの行でなく，1～5 カラムになにかせん孔されていると$カードかどうかの判定を受けて，そうであれば前述のような編集を受け，$カードでなければ，せん孔されているものはステートメントナンバーと見なしてそのチェックを行なう．それが終ると MT に格納してこのルーチンを脱出する．

2.3 STLIST, DOLIST 作成

すでに1ジョブ分のソースプログラムを MT に格納してあるので，チェックされるプログラムユニットのステートメントを一つずつ読み出して来て，ステートメントナンバー表 (STLIST), DO 表 (DOLIST) の作成を行なう．STLIST に登録する際にステートメントナンバーの2重定義も調べておく．STLIST, DOLIST の構造をそれぞれ**第 3, 4 図**に示す．



第3図　STLIST の構造



第4図　DOLIST の構造

第 3, 4 図で a はそのステートメントナンバーあるいは DO ステートメントが存在する位置で，これはそのプログラムユニットの$カードから数えはじめて何番目かで示される．b はステートメントナンバー，C は DO の影響が及ぶ限界を示すステートメントナンバーである．STLIST は 100 語，DOLIST は 50 語分確保してあるので1プログラムユニット中最大 100 個のステートメントナンバー，50 個の DO ステートメントまでがチェック可能である．これらの表を作成する前に最初のカードは$カードかどうか，このプログラムユニットはメインルーチンかどうか調べている．それが$カードでない場合はそのジョブのチェックをそれ以上行なわず次のジョブのチェックに移る．

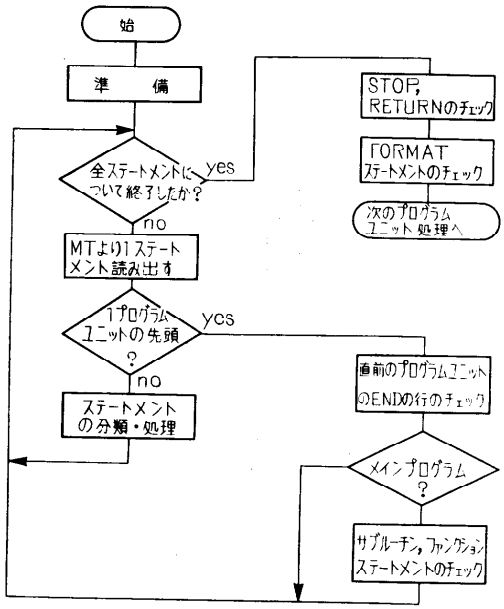
これは，以下の処理でプログラムの暴走を防ぐためである．以上のようにして，当該プログラムユニット内のステートメントナンバー，DO ステートメントを全部調べ終ると制御をチェックルーチンへ移して本格的にチェックを開始する．

2.4 チェックルーチン

MT より1ステートメントずつ読み出してどのようなステートメントか判定し，それぞれのチェックを行なってゆく．ステートメントの分類処理では FORMAT ステートメント，CALL ステートメントをはじめに分離し，次にタイプステートメントが見い出され，integer, complex, array(DIMENSION ステートメントと COMMON ステートメント中に現われる配列名)，logical, real に従って変数名表に登録される．

また READ(5, WRITE(まで見い出されると，ともに RWLIST 作成ルーチンに飛び，あとで FORMAT ステートメント処理の参照に RWLIST を作る (RWLIST については後述)．

GO TO, IF(ではそれぞれ GO TO ステートメント処理，IF ステートメント処理に飛ぶ．END の行が来た時には MT 上の次のブロック内容が$か EOF (MT 上の1ジョブ終了の印) か調べて END の位置が合法かどうかチェックしている．その他，使用禁止



第5図 チェックルーチン

のステートメントかどうか，ステートメントが正しく書かれているかどうか調べ，以上のどれにも該当しなかった場合はそのステートメントに‘＝’が存在するか否かを見て，‘＝’があれば数式と見なし算術論理ステートメントのチェックルーチンに飛ばし，なければ使用しない方がよいステートメントか，HARP 言語にはないステートメントとしてエラー表に登録して次のステートメントのチェックに入る．もし1ジョブ中のプログラムユニットの先頭のステートメントの場合には，直前のプログラムユニットに END の行が存在したかどうかを調べてから，さらにこのプログラムユニットがメインプログラムかどうか判定し，サブプログラムであれば，その先頭のステートメントが，WORD LENGTH n BITS か FUNCTION ステートメントか SUBROUTINE ステートメントになっているか調べ，現れた函数名，サブプログラム名を登録している．全部のステートメントについて終了するとメインプログラムであれば STOP が，サブプログラムであれば RETURN が存在したか調べ，最後に FORMAT ステートメントのチェックを行なって1プログラムユニットの処理を終了する．

2.5 制御ステートメント

（1） GO TO ステートメント

GO TO のあとに‘＝’のないものを GO TO ステートメントと見なし，このルーチンでチェックする．

これをさらに（i）無条件 GO TO (ii) 計算結果による GO TO (iii) 割り当てられた GO TO に判別し，それぞれに変数名とステートメントナンバーの位置，型についてチェックする．飛び越し先がプログラム中に定義されているかを STLIST を用いてチェックし，しかもそれが他の DO の範囲に飛び込んでいないかを DO LIST を用いてチェックするのが GO TO 先チェックの意味である．さらに GO TO の飛び込み先はすべて実行可能なステートメントでなければならないから，これに関するチェックの準備として，飛び越し先ステートメントナンバーを STLIST 中に探し，そのフラグビットをセットしておく．（チェックについては DO ステートメント (iv) 参照）

（2） IF ステートメント

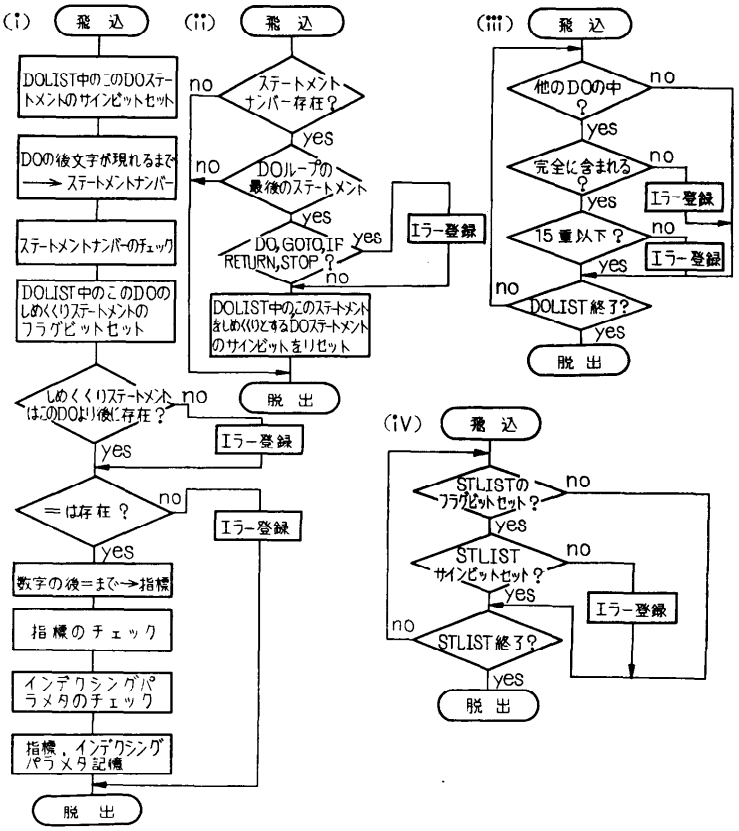
IF(のあとに‘＝’のないものを IF ステートメントと見なし，このルーチンでチェックする．これをさらに最後の‘)’のあとがステートメントナンバーであるか否かで（i）算術 IF (ii) 論理 IF に判別する．

（i） 算術ステートメントの部分は算術ステートメントサブルーチンに飛んでチェックする．最後の‘)’のあとに3個のステートメントナンバーがなければならないものとして読み進む．これは飛び先を意味するから GO TO ステートメントと同様にチェックし，さらに実行可能ステートメントのチェックの準備として STLIST のフラグビットのセットを行なう(チェックは DO ステートメント (iv) 参照)．

（ii） 論理ステートメントの部分は論理ステートメントサブルーチンに飛んでチェックする．最後の‘)’のあとは DO ステートメント，論理ステートメントでないかのチェックを行なうが，これ以外の時それが実行可能であるかのチェックは割愛している．

（3） DO ステートメント

（i） DO の次の文字から最後までをステートメントナンバー，指標（変数名），＝，3（または2）個のインデクシングパラメタ（1個のパラメタは変数名＋定数の形）の各部分に判別し，それぞれを詳細にチェックする．ステートメントナンバーは DO のしめくくりのそれであり，これは実行可能でなければならないから STLIST のフラグビットをセットする．((iv) 参照)．インデクシングパラメタに変数がある時はその変数名（最大3個）と指標（1個）を記憶し，さら



第6図　DO ステートメント

に次のステートメント以後は DO の範囲に入ることを記憶するため，DOLIST 中このステートメントのサインビットセットも行なう．これは DO の範囲内でステートメント函数が定義されていないか，指標，インデクシングパラメタが変えられていないかのチェックのための準備であり，チェックは算術ステートメント，論理ステートメントサブルーチン中で行なう．

指標を記憶する時には現在入っている DO の指標としてすでに使われていないか（指標の2重使用）のチェックを行なう．DOLIST 中のサインビットの解除はメインルーチンで DO の範囲のしめくくりステートメントが現われた時に行なう（(ii) 参照）．

(ii)　メインルーチンでステートメントナンバーのあるステートメントについては DO のしめくくりのものかどうか調べる．そうであれば DO LIST 中のサインビットを解除し，そのステートメントが DO ループのしめくくりのものとして有効かどうかチェックする．

(iii)　メインルーチンで各プログラムユニットごとに DO ループのネスティングのチェックを行なう．DO ループ中の任意の2つの DO は一つの DO の範囲を含む，またはステートメントを共有しない，の関係にあるか，DO の深さは 15 重以下かのチェックである．

(iv)　GO TO ステートメントにおける GO TO 先ステートメント，IF ステートメントにおけるジャンプ先ステートメント，DO ステートメントにおけるしめくくりステートメントなど実行可能であるべきステートメントを STLIST のフラグビットセットによって記憶して来た．一方，メインルーチンでステートメントナンバー付きのステートメントのうち実行可能なものを STLIST のサインビットによって記憶している．そこで STLIST のフラグビットとサインビットとの対照によってプログラムユニットごとに実行可能ステートメントのチェックを行なう．

制御ステートメントにもう一つ CALL ステートメントがあるがこの説明は省略する．

2.6　算術ステートメント

算術および論理ステートメントは，はじめの段階では一つのプログラムでチェックする方法をとっていたが算術式，比較演算子，算術式の形のものを論理式として見分けることが難しく結局同じ方法ではあるが，右辺のチェックを二つのサブルーチンの形式で行なうことにした．このプログラムは大きく分けると左辺のチェック，算術式の右辺のチェック，論理式の右辺のチェックの三つのサブルーチンから成る．このプログラムに入る以前に算術（または論理）式であるとか，ステートメントであるとかが判定され，ステートメントがカード2枚以上にわたる場合には一続きのステートメントとして記憶され，さらにその最後にはステー

トメントの終りを示す記号の入った形でこのプログラムに渡される．

チェックは指定されたカラムから始めて終りの記号が来るまで続ける．チェックの途中でエラーが発見されるとエラーナンバーを記憶し，そこでチェックを打ち切って次のステートメントへ移る．左辺のチェック時に決められる左辺の型によって右辺のチェックに入る時，算術式か論理式のどちらかのルーチンに飛ぶ．そのため IF の括弧の中や CALL の実引数としての算術式や論理式のように左辺のない形のものでは，はじめに現われる変数，定数が論理型であれば左辺の型を論理型として論理式の右辺へ飛び，そうでない場合は左辺の型は未定として算術式の右辺へ飛んでチェックを行なう．

後者の場合にはチェック中に比較演算子が現われると，そこで左辺の型を論理型に修正しその後は論理式としてチェックを行なう．左辺のチェックは変数を見つけ出すことから始める．変数名としてのチェックは文字で始まっているか，6文字を越えてはいないかの二つを調べる．

変数が添字付変数名である時は添字のチェックルーチンに入るが，このルーチンは閉じたサブルーチンで添字の個数が定義と合っているか，型や形は正しいか，などのチェックを行なう．変数が添字付変数名でなく，しかもその後に‘(’が続いて来る時はそのステートメントをステートメント函数の定義と見なす．左辺のチェックプログラムはサブルーチンの形式をとり，‘二’までチェックすると一度そとに飛び出し，改めて左辺の型を判定して右辺のチェックに移る．右辺もサブルーチンの形式をとるが，これは前にも述べたが，IF の括弧の中のような右辺だけの形のものもこのプログラムでチェックするためである．左辺が論理型の時の右辺は論理ステートメントの所で述べる．

次に右辺に現われる括弧であるが，これは演算の順位を上げるためのもの，函数または添字付変数に続いて現われるもの，複素定数の括弧の三つに分けられる．函数の実引数のチェックも右辺の中で行なうため実引数として函数が現われるような場合の函数の重なりの深さを考え‘(’や‘)’がどの深さに属するかを分類し，それによって引数がどの函数のものであるかを調べている．そのために一見不要と思われる函数の重複度を数えている．

添字付変数，すでに現われた函数名以外の変数で，あとに‘(’が続いて来る時，それを函数サブプログラム名として記憶する．‘·’または数字が現われた場合には定数の判定をする．‘·’の場合には次に比較演算子があれば算術式・比較演算子の形であることを記憶して再び算術式のチェックに入る．比較演算子がない時は，もしそれ以前に比較演算子が来ていれば算術式・比較演算子・算術式の形であると判断して論理式のチェックに入る．この時左辺の型が未定であれば論理型になおして論理式のルーチンに飛ぶ．

数字の場合には次に現われる数字でないものが‘·’であれば実数定数，H であれば文字型定数，それ以外の場合は整数定数と判定する．実数定数と判定されると小数点以下や指数があるかなどを調べて，正しければさらにそれが複素定数であるかどうか調べる．‘)’の処理はその時の函数の深さによって分類し数える．その‘)’によってちょうど閉じるような函数があればそこで函数の深さが一つ下る．‘,’が現われた場合，それが函数の中であればその時の函数の深さによってその函数の実引数の数を一つ増す．ステートメントが終りにならない限り変数，函数，定数のあとは‘)’か演算子が，‘)’のあとは演算子が，演算子のあとは変数，函数，定数，‘)’のどれかが現われねばならない．ステートメントの終りの記号が検出された時，または指定されたカラムまでチェックを終った時，最後のしめくくりとして，括弧が全部閉じているか，左辺と右辺は許される型であるかを調べて算術ステートメントのチェックを終る．

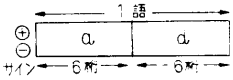
2.7 論理ステートメント

算術ステートメントの項で述べたように，このルーチンは論理式として正しいかどうかをチェックするものである．方法は算術ステートメントと全く同じであるが，型の判定としては論理型かどうかを調べるだけである．定数としては TRUE と FALSE のみで，演算子としては OR と AND と NOT があるが，算術式で幅乗を特別に扱っているように NOT は注意して扱っている．

2.8 入出力および FORMAT ステートメント

READ(5 あるいは WRITE(まで見い出されると，このルーチンに飛んで来る．もし，このステートメントが)= の形をしていたら数式処理へ飛ばす．そうでなければ，このルーチンで‘(’と‘)’との一致や機番指定，FORMAT 指定などについてチェックし，FORMAT ステートメントのチェックに役立てるた

めに RWLIST を作成する．これは**第7図**に示すようなもので，サインビット⊕は読み取り，⊖は書き込



第7図　RWLIST の構造

みの場合である．aはそのステートメントが存在する位置を示し，dはそれが参照するFORMATのステートメントナンバーである．

次に FORMAT ステートメントのチェックは，はじめにステートメントナンバーが存在するかどうか調べ，登録されていなければ数式かどうか調べられる．)= の形を持たず，FORMAT(の形をしていたら，これは FORMAT ステートメントと考える．一方 RWLIST を見てこの FORMAT は READ に対するものか WRITE に対するものかを決定する．これは D, E, F 要素の処理で必要になる．

次に，一字ずつ順に調べられた反復指定，巾因子の前の整数定数が分離され，以下各要素，巾因子の処理を行なう．

以上のどれにも該当しなかった－，/,(などは最後に処理されて，以後は全く前と同様に次の文字から調べてゆく．整数定数処理では数字が尽きるまで読み，数字モードに直して記憶しておく．これは H 要素，巾因子の処理の時に役立てる．

D, E, F 要素の処理では w·d の形が合法かどうかをチェックする．H 要素では WRITE に対応する FORMAT であって '(' あるいは '/' のあとの H 要素に制御用文字が積極的に書かれているか調べるほかは，前述の整数定数処理で得られた字数だけ読み飛ばしている．次の ',' はなくてもよいので注意を要する．

I, L, A, O, X 要素ではあとに述べること以外はチェックしない．

巾因子の処理では尺度を示す整数定数が8以下かを調べる．各要素において必ず次の要素が始まる直前の文字までをチェックするので各要素，巾因子，－，/, (，ステートメント終了コードのどれにも該当しない時は，記号を不当に使ったものと見なされ，以後はチェックせず次の FORMAT ステートメントに進む．

3. 実　　例

ここに示した実例のチェックに要する時間は，約1分20秒であった．

```
 1 $SIMPSON   HARP
   C      SIMPSON NI YORU TEISEKIBUN
 2        READ(5,100) A,B
 3   100  FORMAT(2E15.7)
 4        N=2
 5        MULT=(B-A)/3.0
 6        STEP=0.5*(B-A)
 7        X=A
 8        CAL FUNC(X)
 9        K=0.5*(K+F)
10  1000  CONTINUE
11        SBAR=K*MULT
12        CBAR=1
13      1 J=0.0
14        X=A+FLOAT(N)*STEP
15        J=J+F/N
16        DO1000 L=1,2,N
17        I=K+4.0*J
18        SIMPS=I*MULT
19        WRITE(6,200) SIMPS
20        PAUSE
21        IF(ABS((SIMPS-SBAR)/SIMPS).GT.10**(-7)) GO TO2
22        C=0
23        GO TO3
24      2 C=1
25        IF(C.NE.0) GO TO5
26      5 N=2*N
27        STEP=0.5*STEP
28        K=J+0.5*K
29        SBAR=SIMPS
30      2 CBAR=C
31        GO TO 1
32        END

 DECK NAME     LINE NUMBER     ERROR NUMBER

  SIMP             30             7005
  SIMP              8             8000
  SIMP              9             1701
  SIMP             15             1701
  SIMP             16             3503
  SIMP             17             1701
  SIMP             19             4005
  SIMP             20             9000
  SIMP             23             3003
  SIMP             28             1701
  SIMP             32             3400

              CHECK END
```

第8図

4. 結　　言

このプログラムで相当程度のエラーの検出は可能であるが，より完全なものに近づくために改良すべき点を幾つか述べてみる．

エラーナンバー表

コントロールカード	
0	$HARP, $ENTRY でない $DATA, $LOADER
1	デック名の始めの4文字空白
2	$ENTRY カードが許されない位置にある
算術ステートメント	
1100	変数名，函数名が7文字以上である
1101	変数名，函数名の始めの字が数字である
1102	変数名がない
1200	文字型定数の誤り
1202	E.Dのあとに数字がない
1203	指数が39以上
1204	複素定数の誤り
1205	複素定数の）がない
1300	添字が整数型でない
1301	添字として不適当な記号がある
1302	添字の個数が定義と異なる 添字付変数のあとに（がない
1400	（と）との現われ方が適当でない
1500	演算子がない
1600	函数名のあとに（がない
1601	助変数の個数が17以上
1602	函数名が左辺に現われた
1603	ステートメント函数の定義の助変数の個数が17以上
1604	実行可能ステートメントよりあとで函数が定義されている
1700	**のあとの型が適当でない
1701	演算子の左右の型が適当でない
1702	等号の左右の型が適当でない
1900	・が不適当な所に現われた
1901	左辺に不適当な記号がある
1902	右辺に不適当な記号がある
論理ステートメント	
2100	変数名，函数名が7文字以上である
2101	変数名，函数名の始めの字が数字である
2102	変数名がない
2200	文字型定数の誤り
2202	E.Dのあとに数字がない
2203	指数が39以上
2250	FALSE, TRUE のあとに・がない
2300	添字が整数型でない
2301	添字として不適当な記号がある
2302	添字の個数が定義と異なる
2303	添字付変数名のあとに（がない
2400	（と）との現われ方が適当でない
2500	論理式中の算術式に演算子がない
2550	NOT のあとに・がない
2551	論理演算子がない
2600	函数名のあとに（がない
2601	助変数の個数が17以上
2602	函数名が左辺にあらわれた
2603	ステートメント函数の定義の助変数の個数が17以上
2604	実行可能ステートメントよりあとで函数が定義されている
2700	**のあとの型が適当でない
2702	右辺が論理型でない
2901	左辺に不適当な記号がある
2950	右辺に不適当な記号がある
制御ステートメント	
3000	ステートメントナンバーなし
3001	ステートメントナンバーが数字でない
3002	ステートメントナンバーが6文字以上
3003	ステートメントナンバーの未定義
3004	GOTO 先のステートメントが実行不可能なステートメント
3005	（がない
3006	）がない
3007	・がない
3008	変数名がない
3009	変数名の先頭が文字でない。
3010	変数名が7文字以上
3011	変数名が整数型でない
3012	変数名が他ですでに使われている
3013	変数名に文字，数字でないものがある
3014	DOループに飛込んでいる
3201	3002に同じ
3202	3000に同じ
3203	3001に同じ
3204	論理IFの中のステートメントの違反
3400	メインプログラムユニットにSTOPがない
3500	3002に同じ
3501	3000に同じ
3502	3003に同じ
3503	DOループの最後のステートメントがDOループより前にある
3504	DOループの最後のステートメント違反
3510	＝がない
3511	3009に同じ
3512	3010に同じ
3513	3013に同じ
3514	3008に同じ
3515	インデックスが多すぎる
3516	インデックス不足
3517	Cが整数型定数でない
3518	Cが32767より大きい
3519	Cが正でない
3520	n_1がn_2より大きい
3530	DOループの中でインデックスが変換されている
3531	DOループの重なりが無効
3532	DOループの重なりが16重以上
3533	指標の2重使用
3700	CONTINUE の誤り
3801	チェック中止
3802	サブルーチン名が7文字以上
3803	（がない
3804	）がない
3805	サブルーチン名がない
3806	サブルーチン名の先頭が文字でない
3807	サブルーチン名が他ですでに使われている
3808	対応するサブルーチンサブプログラムがない
3900	サブプログラムユニットにRETURNがない
3901	メインプログラムユニットにRETURNがあった
3902	RETURN の誤り
入出力ステートメント	
4001	（と）との数が合わない
4002	WRITE ステートメントの機番の指定の誤り
4003	誤せん孔
4004	FORMATを示す1次元配列名がない
4005	FORMAT を指定するステートメントナンバーが定義されてない
4006	）がない
4010	FORMAT ステートメントの対応する入出力ステートメントがない
4011	WRITEのFORMATで制御用文字が積極的に用いられていない
4012	区切り符号・/あるいは）がない
4013	D，E要素において$w \geq d+7$でないか・がない
4014	F要素において$w \geq d+3$でないか・がない
4015	nP, nQのnが$-8 \leq n \leq 8$でない
4016	（と）との数が合わない
4017	記号の不当使用
4018	…,/…となっている
4019	…,）…となっている
4020	記号の不当使用
仕様ステートメント	
5000	DIMENSION, … COMMON, …とした．
5001	END の行がない
5002	END の行の位置が違う
5003	DIMENSION, COMMON 中の配列宣言子の誤り
5004	INTEGER のタイプステートメントの誤り
5005	COMPLEX のタイプステートメントの誤り
5006	LOGICAL のタイプステートメントの誤り
5007	配列宣言，タイプステートメントで変数名が7文字以上
5008	4次元以上の配列宣言
5009	EQUIVALENCE の誤り
5010	EXTERNAL の誤り
5011	2倍精度の誤り
5012	DOUBLE PRECISION の誤り
5013	4倍精度の誤り
5014	ステートメントの記述順序の誤り
5015	COMMON のブロック名エラー
サブプログラム 函数ステートメント	
6001	CALLに現われた函数名，サブプログラム名がない
6003	函数，サブプログラムステートメントの誤り
6004	サブプログラム名の誤り
6005	サブプログラムに函数，サブプログラムステートメントがない
その他	
7001	コンティニュエーションの行，10行を越えた
7002	カラム6に数字以外のものをせん孔
7003	カラム6の数字が数上りでない
7004	語長指定の誤り
7005	ステートメントナンバーの2重定義
7010	ステートメントナンバー制限範囲外
7011	コンティニュエーションの行の1～5カラムに何かせん孔されている
8000	用がない方が望ましいステートメントかHARPに許されないステートメント
使用禁止	
9000	PAUSE である
9001	IF(SENSE LIGHT i) n_1, n_2である
9002	IF(SENSE SWITCH i) n_1, n_2である
9003	ASSIGNである
9004	SENSE LIGHT kである
9005	READ(k) リストである
9007	READ n, リストである
9008	PUNCH n, リストである
9009	PRINT n, リストである
9010	READ TAPE k, リストである
9011	WRITE TAPE k, リストである
9999	エラーの総数が50を越えた

はじめにチェックプログラム本体を数ブロックに分割して MT に入れておき順次呼び出すようにすることである．このようにすれば種々の表がコア内に比較的大きくとれ相当詳しいチェックが可能になる．諸表の作り方もビット単位で作るべきであった．現在は計算機固有のコードを用いて種々の表を作成しているので手間，スピードの点ではよいのであるが無駄が多い．完全を目指すならば，変換ルーチンを通してビット単位の表を利用せねばならない．ステートメントの分離については，算術式の左辺の資格を満たすかどうかで早く算術式を分離してしまう方がよいように思われる．最後までこのステートメントはまだ算術式の可能性があると考えねばならないのは非常に面倒であった．

以上のように至らぬ点は数多くあるが，今回の研究活動により，得ることの大きかったことに対して，2年間お世話いただいた方々に感謝の意を表したい．

6. エラーナンバー表

274 ページを参照．

（昭和42年4月4日受付）