ANL-7628

# Argonne National Laboratory

## A SYMBOLIC DERIVATIVE-TAKER FOR A SPECIAL-PURPOSE LANGUAGE FOR LEAST-SQUARES FITS

by

Marian Gabriel

The facilities of Argonne National Laboratory are owned by the United States Government. Under the terms of a contract (W-31-109-Eng-38) between the U. S. Atomic Energy Commission, Argonne Universities Association and The University of Chicago, the University employs the staff and operates the Laboratory in accordance with policies and programs formulated, approved and reviewed by the Association.

MEMBERS OF ARGONNE UNIVERSITIES ASSOCIATION

The University of Arizona
Carnegie-Mellon University
Case Western Reserve University
The University of Chicago
University of Cincinnati
Illinois Institute of Technology
University of Illinois
Indiana University
Iowa State University
The University of Iowa
Kansas State University
The University of Kansas
Loyola University
Marquette University
Michigan State University
The University of Michigan
University of Minnesota
University of Missouri
Northwestern University
University of Notre Dame
The Ohio State University
Ohio University
The Pennsylvania State University
Purdue University
Saint Louis University
Southern Illinois University
University of Texas
Washington University
Wayne State University
The University of Wisconsin

## LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

Printed in the United States of America
Available from
Clearinghouse for Federal Scientific and Technical Information
National Bureau of Standards, U. S. Department of Commerce
Springfield, Virginia 22151
Price: Printed Copy $3.00; Microfiche $0.65

ANL-7628
Mathematics and Computers

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

# A SYMBOLIC DERIVATIVE-TAKER FOR A SPECIAL-PURPOSE LANGUAGE FOR LEAST-SQUARES FITS

by

Marian Gabriel

Applied Mathematics Division

February 1970

# TABLE OF CONTENTS


| | Page |
|---|---|
| ABSTRACT | 5 |
| I. BASIC CONSIDERATIONS | 5 |
| II. THE SCANNING ROUTINES | 6 |
| III. TRIPLE-HANDLING ROUTINES | 8 |
| IV. DERIVATIVE-TAKING ROUTINES | 11 |
| V. SIMPLIFICATION | 12 |
| VI. CONCLUSIONS | 13 |
| APPENDIXES | |
| A. Special-purpose Language for Least-squares Fits | 14 |
| B. Flow Charts for the Derivative-taker | 25 |
| C. Listing of DERV | 56 |
| D. Listings of Cataloged Procedures VMMCLG and VMMLG | 69 |
| E. Sample Function and Its Derivatives | 71 |
| ACKNOWLEDGMENT | 72 |
| REFERENCES | 72 |

## LIST OF FIGURES


| No. | Title | Page |
|---|---|---|
| 1. | Production for A**B | 9 |
| 2. | Redefinition of Pointers after a Triple Is Dropped. | 9 |
| 3. | Sorting of Triples by SC Value. | 10 |
| 4. | Replacement of Constants in the Expression for the Derivative | 11 |
| B1. | Flow Chart 1, ANALYS. | 25 |
| B2. | Flow Chart 2, GTNAM | 28 |
| B3. | Flow Chart 3, BACKUS. | 31 |
| B4. | Flow Chart 4, MKTRPL. | 36 |
| B5. | Flow Chart 5, STPTRS. | 37 |
| B6. | Flow Chart 6, SHRINK. | 38 |
| B7. | Flow Chart 7, SORT. | 40 |
| B8. | Flow Chart 8, STORE | 41 |
| B9. | Flow Chart 9, REP | 42 |
| B10. | Flow Chart 10, DRV Outline. | 44 |
| B11. | Flow Chart 11, DRV(ST,X). | 45 |
| B12. | Flow Chart 12, RPLCNS(STR). | 47 |
| B13. | Flow Chart 13, SIMP Outline | 49 |
| B14. | Flow Chart 14, SIMP | 50 |
| B15. | Flow Chart 15, CKOPN. | 54 |


## LIST OF TABLES


| No. | Title | Page |
|---|---|---|
| I. | Operation of ANALYS (First Scan) on a Sample String | 6 |
| II. | Operation of ANALYS (Second Scan) to a Sample String. | 6 |
| III. | Arrays ELT and SIG for a Sample String. | 7 |
| IV. | Array BNF for A**B. | 7 |
| V. | Formation of the Production for A**B. | 8 |
| A.I. | Symbolic Parameters and Default Values for VMMCLG and VMMLG | 17 |

# A SYMBOLIC DERIVATIVE-TAKER FOR A SPECIAL-PURPOSE LANGUAGE FOR LEAST-SQUARES FITS

by

Marian Gabriel

ABSTRACT

This report describes a computer program for performing symbolic differentiation of FORTRAN formulas. The process by which formulas are analyzed, their analytic derivatives found, and the resulting derivative expressions simplified is described in detail. A program listing and a sample output are included.

## I. BASIC CONSIDERATIONS

This report describes a program that analyzes functions, defined by FORTRAN formulas, and produces FORTRAN expressions for their derivatives. The program was written to be a part of a special-purpose language for doing least-squares fits.[1] The curve-fitting program used Davidon's method,[2] which requires partial derivatives of the function to be fitted, and thus the user had to supply FORTRAN expressions for these derivatives. Finding them for a complicated function is, at best, a nuisance. At worst, it is a common source of errors. It seemed desirable, therefore, to add a symbolic derivative-taker to the language. Within this language, the derivative-taker may be called by using the statement DPNAME=DERV, where PNAME is the name of a fitting parameter. (See Appendix A for a full description of the language.)

During the entire process of writing this special-purpose language, the emphasis has been on producing, as quickly as possible, reasonable programs that would be easy to use. No significant effort was directed toward making either the translators or the generated code maximally efficient. In writing the derivative-taker, the methods and language used were those that happened to be available. The programming was done in PL/I; text-scanning was done by the method given by Sheridan.[3]

Since Sheridan's methods for handling user-defined functions and arrays are oriented more to assembly language than to PL/I, these two types of functions were excluded from the derivative-taker capabilities. These restrictions may be removed in the future.

This description of the derivative-taking program is divided into the following sections: scanning routines, triple-handling routines, derivative-taking routines, and simplification routines. Only a small part of the program, and of the description, is concerned with actually evaluating the derivative according to the rules of calculus. Most of the work lies in analyzing the structure of the FORTRAN formula so that these rules may be applied.

## II. THE SCANNING ROUTINES

The function whose derivative is required is passed to the derivative-taker by a statement of the form D=DERV(FCN,X). Here FCN is a string of at most 1320 characters; X is a string of at most 6 characters.

First, DERV must separate this string into an array of names, constants, and operators, and indicate the hierarchy of the operations. Two routines, ANALYS and GTNAM, make a list of names, constants, and operators. The first of these, ANALYS, sets up arrays of bits to describe the characters of the string. ANALYS scans the string twice. On the first scan one of the three bits OPP(I), ALP(I), and NUM(I) is set to '1' (or "true" or "on"), depending on whether the $I^{th}$ character is an operator, a letter, or a digit or decimal point, respectively. The other two bits for the particular I are set to '0'. For instance, for the string 5.2E+06*EXP((X/Y)**2.) + X1, Table I gives the values of OPP, ALP, and NUM for each character.

TABLE I. Operation of ANALYS (First Scan) on a Sample String

| I | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CHARACTER | 5 | . | 2 | E | + | 0 | 6 | * | E | X | P | ( | ( | X | / | Y | ) | * | * | 2 | . | ) | + | X | 1 |
| OPP | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| ALP | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| NUM | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

On the second scan, one of the bits OP(I), VAR(I), or CONST(I) is set to '1' to indicate whether the $I^{th}$ character is an operator, part of the name of either a variable or a library function, or part of a number. This is done by examining the first character of the string following each operator. If the character is a letter, the string is a name; otherwise, it is a number. Table II gives the results of applying these rules to the sample string above. Here the characters 'E+' are treated as part of the number. Figure B1 of Appendix B describes ANALYS in flow chart form.

TABLE II. Operation of ANALYS (Second Scan) to a Sample String

| I | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CHARACTER | 5 | . | 2 | E | + | 0 | 6 | * | E | X | P | ( | ( | X | / | Y | ) | * | * | 2 | . | ) | + | X | 1 |
| OP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| VAR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| CONST | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

GTNAM assigns a name to each constant in the string S (FCN with blanks removed) and expresses S as an array ELT, where each member of ELT is a name, an operator, or a name representing a constant. GTNAM also stores the constants in an array CONS and defines an array, SIG, of bits by setting SIG(I) to '1' if ELT(I) is a name or a left parenthesis and to '0' otherwise. For the string above, the members of ELT and SIG would be as shown in Table III. The array CONS would have two elements, 5.2E+06 and 2. Figure B2 shows the flow chart for GTNAM.

TABLE III. Arrays ELT and SIG for a Sample String

| I | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ELT(I) | C$1 | * | EXP | ( | ( | X | / | Y | ) | ** | C$2 | ) | + | X1 |
| SIG(I) | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

BACKUS, the final routine in this section, creates the Backus normal form, which indicates the priority of operations in the string. The routine uses the method described by Sheridan.[3] The Backus normal form indicates the "separation" between two operands by adding extra parentheses and operations. For instance, in a FORTRAN expression, the additions are performed last. Therefore, the operator '+' "strongly separates" two variables, and A+B becomes

+(*(**(#A))) + (*(**(#B))) ('#' represents the action of a function).

In contrast, '**' indicates a "weak separation." The Backus normal form of A**B is

+(*(**(#A)**(#B))).

For convenience in programming the next sections, the program BACKUS returns a two-dimensional array BNF with two columns. For a given I, either

(a) BNF(I,1) is null and BNF(I,2) = ')',

or

(b) BNF(I,1) is '+', '-', '*', '/', '**', or '#' and BNF(I,2) is '(' or a name.

The array BNF created by BACKUS for the string A**B is shown in Table IV.

TABLE IV. Array BNF for A**B

| I,J | 1,1 | 1,2 | 2,1 | 2,2 | 3,1 | 3,2 | 4,1 | 4,2 | 5,1 | 5,2 | 6,1 | 6,2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BNF(I,J) | + | ( | * | ( | ** | ( | # | A | null | ) | ** | ( |

| I,J | 7,1 | 7,2 | 8,1 | 8,2 | 9,1 | 9,2 | 10,1 | 10,2 |
|---|---|---|---|---|---|---|---|---|
| BNF(I,J) | # | B | null | ) | null | ) | null | ) |

Further details of BACKUS are given in Fig. B3.

## III. TRIPLE-HANDLING ROUTINES

Once BNF has been established for the string, a number is assigned to each nonnull operation (BNF(I,1)) and to the operand (BNF(I,2)) following it. This number depends on the number of left and right parentheses preceding the operation in the sequence of operations in the string. These numbers are calculated by counting left and right parentheses and constructing a list of triples, called a *production*, where each triple contains a number, an operation, and an operand. When all the numbers have been assigned, the triples corresponding to extraneous operations in the Backus normal form are dropped, and the remaining triples are listed in order of increasing number.

PL/I list-processing facilities are used extensively in this section. Each triple is represented by a based structure TRPL that contains the number, SC; the operation, DI; the operand, PSI; and a bit string SV. TRIPL also contains two pointers (UP and DOWN) to define its position in the production.

The first routine in this section, MKTRPL, allocates a triple for each nonnull BNF(I,1), sets SV to '0', and defines SC, DI, and PSI by the following method. Let N, A, and C be integers, and let K be a sequence of integers. As the Backus normal form is scanned from left to right, N gives the total number of left parentheses that have occurred, A gives the difference between the number of left parentheses and the number of right parentheses, C gives the hierarchy of the current operation, and K is a list of the previous values of C. Initially, N = 1, both A and C are zero, and K has no elements. For each I, SC, DI, PSI, then, N, A, C, and K are defined inductively by BNF(I,1) and BNF(I,2) according to the following rules:

1. If BNF(I,2) is a left parenthesis, SC = C, DI = BNF(I,1), and PSI = N. Now K becomes K $\cup$ {C}; C is set to N; and N and A are each increased by 1.

2. If BNF(I,2) is a name, SC = C, DI = BNF(I,1), and PSI = BNF(I,2). Then K, C, N, and A all remain the same.

3. If BNF(I,2) is a right parenthesis, BNF(I,1) is null, and TRPL is not allocated. K becomes K - $\{C_n\}$, where $C_n$ is the last C value added to K. C becomes $C_n$, and A is decreased by 1. N remains fixed. Table V illustrates the formation of the production for A**B.

TABLE V. Formation of the Production for A**B

(SC, DI, PSI, N, C, A, and K values are determined by the BNF values in the preceding column; "Λ" indicates no value.)

| BNF(I,1) BNF(I,2) | +( | *( | **( | #A | ) | **( | #B | ) | ) | ) | Λ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SC | Λ | 0 | 1 | 2 | 3 | Λ | 2 | 4 | Λ | Λ | Λ |
| DI | Λ | + | * | ** | # | Λ | ** | # | Λ | Λ | Λ |
| PSI | Λ | 1 | 2 | 3 | A | Λ | 4 | B | Λ | Λ | Λ |
| N | 1 | 2 | 3 | 4 | 4 | 4 | 5 | 5 | 5 | 5 | 5 |
| C | 0 | 1 | 2 | 3 | 3 | 2 | 4 | 4 | 2 | 1 | 0 |
| A | 0 | 1 | 2 | 3 | 3 | 2 | 3 | 3 | 2 | 1 | 0 |
| K | Λ | (0) | (0,1) | (0,1,2) | (0,1,2) | (0,1) | (0,1,2) | (0,1,2) | (0,1) | (0) | Λ |

MKTRPL also calls STPTRS to set UP and DOWN. UP is set to null for the first triple and points to the preceding triple for all others. DOWN is null for the last triple and points to the succeeding triple for all others. Another pointer, HEAD, points to the first triple in the list; TAIL points to the last. For example, at the end of MKTRPL, the production for A**B would look as shown in Fig. 1. Since all values of SV are '0', they are omitted from the diagram. Figures B4 and B5 describe MKTRPL.

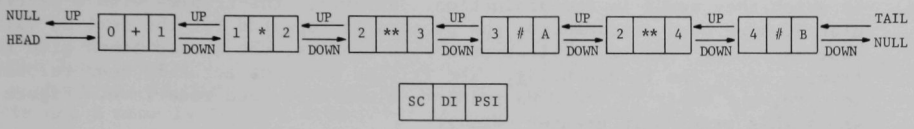



Fig. 1. Production for A**B

The next routine, SHRINK, scans the list, starting with the triple at TAIL, and drops any triple whose SC value appears only once in the production, unless the DI value of the triple is '-'. When a triple is dropped, its PSI value replaces the PSI of the triple immediately preceding it.

Two pointers, PT1 and PT2, are used to scan the list. PT1 moves from triple to triple, starting with TAIL and ending with HEAD. If PT1 → SV* is '1', PT1 is replaced by UP, so that PT1 now points to the preceding triple. Otherwise, PT2 moves from PT1 to HEAD.

If PT1 → SV is '0', then the production is searched for SC values equal to PT1 → SC. This is done by making PT2 progress through the production, beginning with PT1 and ending with HEAD.

For each value of PT2, a comparison of PT2 → SC is made with PT1 → SC; if they are equal, both PT2 → SV and PT1 → SV are set to '1'. If no PT2 → SC equals PT1 → SC, the entire triple addressed by PT1 is dropped unless PT1 → DI is '1'. When a triple is dropped, the pointers UP and DOWN are redefined so that the list structure is preserved (see Fig. 2). Here the middle triple is to be dropped. The dotted lines represent the new values of the pointers UP and DOWN. SHRINK is described in Fig. B6.

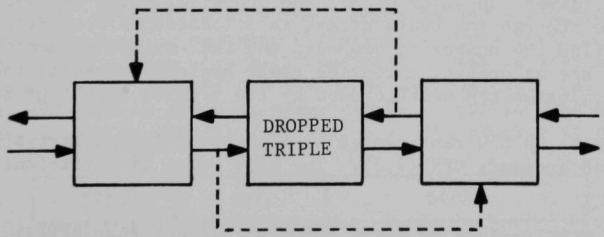



Fig. 2. Redefinition of Pointers after a Triple Is Dropped

*The notation 'PT1 → SV' denotes the SV of the triple to which PT1 points.

Next, SORT arranges the triples in order of increasing SC. This sorting method depends on the fact that when a given SC value first appears in the list, it is greater than any SC preceding it. Thus, additional triples with the same SC can simply be placed behind the first one.

Always HEAD has an SC value of 0. The production is searched for other triples with SC = 0, which are placed immediately behind HEAD in the order in which they occur in the production. After all the triples with a particular SC have been resequenced, the process is repeated for the next SC value that is found. Figure 3 illustrates this process; the dotted lines give the triples after the resequencing. The triples have not actually been relocated in core, but their UP and DOWN pointer values have been redefined. Figure B7 shows this process in greater detail.

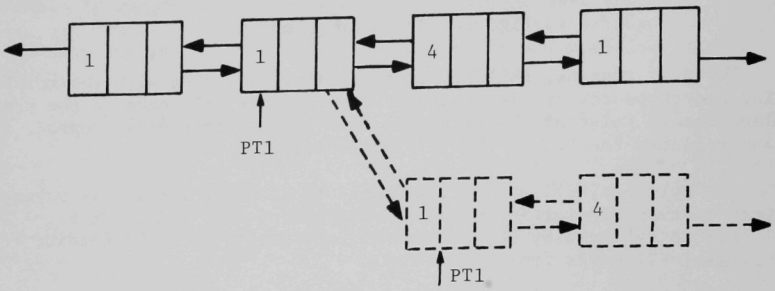



Fig. 3. Sorting of Triples by SC Value (PT1 is the search pointer)

At this point, the reduced production may be organized as a list of segments, where a segment is the list of all triples with the same SC value. The operations in a segment are all performed at the same time in the evaluation of the string. Because the PL/I list-processing facilities were not available when these derivative-taking routines were written, the following arrays are constructed for the segments. The segments are numbered in order of increasing SC, and S(I) is defined as the SC value of the $I^{th}$ segment. LS(I) gives the number of triples in the $I^{th}$ segment; OPR(I,J), for J = 1, ..., LS(I), gives the value of DI; and OPND(I,J) gives the values of PSI. Before core storage for these arrays is allocated, a procedure COUNT is called to find the number of segments and the maximum segment length. Once the arrays are allocated, a routine STORE assigns values to the array elements and releases the storage used by the triples (see Fig. B8).

A segment is now represented as a number S(I) and arrays of operations OPR(I,J) and operands OPND(I,J). The S(I) value is sufficient to identify a segment.

Last, CKRF defines an array RFX of bits such that RFX(I) is '1' if for some J, OPND(I,J) = X or OPND(I,J) equals some segment S(K) such that RFX(K) is '1'. Otherwise RFX(I) is '0'. This routine detects segments that have no references to X, and therefore do not need to be further analyzed.

## IV. DERIVATIVE-TAKING ROUTINES

Three routines are used for the actual derivative taking: REP, which represents a variable or segment as a character string; DRV, which gives the derivative of a segment; and RPLCNS, which replaces the symbols C$I by the actual constants. Both DRV and REP are recursive procedures. REP is straightforward, and can be understood from its flow chart (see Fig. B9) and its listing in Appendix C.

DRV solves the following problems. The routine examines sequences of the form <operand> <operator> <operand> and expresses the derivative of each such string in terms of the derivatives of the operands. Each operand that is not a name is further decomposed into a similar sequence. If an operand is a name, it is compared to X; then if the operand name is X, its derivative is set to 1.0, and otherwise the derivative is set to 0.0.

When a formula is evaluated, the first operations performed are in the segment N with the largest S(I) value. Within a segment, operations other than exponentiation are performed from left to right; exponentiation is performed from right to left.

In DRV this order is exactly reversed. Derivatives are taken starting with the first segment. If a segment contains no references to X (i.e., RFX(I) is '0'), no analysis is done, and the string '0.0' is returned for the derivative. Within a segment the rightmost operation, except for '**', is examined first. Each time DRV operates on a segment, the segment length is reduced by 1. In a segment of length 1, the operand (OPND(I,1)) replaces the segment in the next call of DRV. When the operation is '**', the operands OPND(I,1) and OPND(I,2) are examined first, and the first operand and operation are dropped first.

When the operation is '#' (the operation of a function), provision is made only for a FORTRAN library function with one argument, that is, a segment of length 2. A routine FNDFCN searches the list of these functions and returns the appropriate derivatives. See Figs. B10 and B11 for the details of DRV.

DRV returns a character string that gives the FORTRAN expression for the derivative, but with the constants represented as C$I, where I is a positive integer. RPLCNS replaces these names by the actual constants. The replacement is done by scanning the string for the first occurrence of the characters 'C$'. When they are found, the string is divided into three sections: STRA is the part of the string before the C$I. C$I is the second section, and STRB is the part after the C$I. C$I is then replaced by the entry designated by CONS(I) (see Fig. 4). The process is repeated until

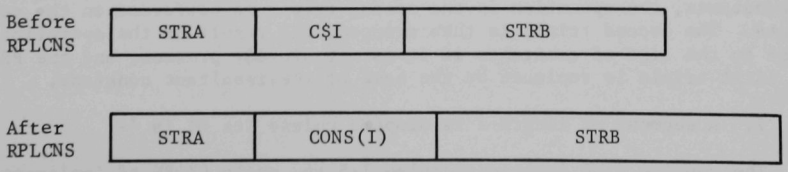



Fig. 4. Replacement of Constants in the Expression for the Derivative

all occurrences of C$I have been replaced by appropriate constants. The details of this process are shown in Fig. B12.

## V. SIMPLIFICATION

RPLCNS returns a string DSTR, which constitutes a FORTRAN formula for the derivative of FCN with respect to X. Even though segments containing no references to X were not analyzed, this formula contains an enormous number of extraneous operations, such as multiplications by zero and one. SIMP and the routines called by it eliminate these operations.

First, DSTR is represented as a list of sorted triples, as described in Sections II and III. Then triples or groups of triples are dropped to eliminate most of the extra operations. When a segment is dropped, its SC value is eliminated from the production, and all PSI's that refer to that SC must be replaced.

The following rules determine which triples or segments to drop:

1. A triple with DI = '+' or '-' and PSI representing 0.0 (i.e., of the form C$I, where CONS(I) = 0.0) is dropped. If the triple forms a segment of length 1, PSI's that refer to the segment are replaced by the "name" of 0.0.

2. A segment that contains a triple with DI = '*' and PSI representing 0.0 is dropped. References are replaced by the "name" of 0.0.

3. If a triple has DI = '**' and PSI representing 0.0, it and the part of the segment following it are dropped. If it is the first triple in the segment, PSI references to the segment are replaced by the "name" of 0.0. If it is not the first triple, the PSI of the triple immediately preceding it is replaced by the name of the constant 1.0, since A**0.0 = 1.0.

4. A triple with DI = '/' and PSI = the name of the constant 1.0 is dropped. A triple with DI = '*' and PSI the name of the constant 1.0 is dropped unless it is immediately followed by a triple in the same segment with DI = '/'. In the latter case, the triples represent the operation 1.0/A, where A may be a variable, a constant, or a formula (given by another segment).

5. A triple with DI = '**' and PSI = the name of the constant 1.0 is dropped. If it is in a segment of length 1, PSI references to the segment are replaced by the "name" of 1.0.

6. If two triples in the same segment have PSI's equal to the names of constants, the operation in the second triple is performed on the constants. The second triple is then dropped, the result of the operation is added to the list of constants if it is not already present, and the PSI of the first triple is replaced by the name of the resultant constant.

7. A segment of length 1 is dropped unless its DI is '-'.

The routine SIMP implements rules 1-5 and calls CKOPN to implement 6 and CKLN to implement 7. In SIMP, the triples are scanned, starting with

the last one. Scanning is repeated until a scan occurs in which no triples are dropped. The first segment in the list is never dropped. Figures B13 and B14 describe SIMP; Fig. B15 describes CKOPN. CKLN is described by the listing in Appendix C.

The following "service routines" are also used to implement the above rates:

1. DTRPL drops a single triple, as determined by rule 1, 4, or 7.

2. DRST implements rule 3 by dropping all of the segment after the occurrence of a '**0.0' operation. It is also used for rule 5 when the '**1.0' operation is not the first triple in segment.

3. RPLRF replaces PSI references to a dropped segment by the name of the constant that replaces the segment.

4. DSG drops segments containing '*0.0' (rule 2), and segments containing '**1.0' as the first triple (rule 5).

5. MKCONS(NUM) searches the list of constants for an occurrence of the constant NUM. If there is some constant CONS(I) which is equal to NUM, the routine returns the character string C$I. If there is no such CONS(I), the number of constants NCONS is increased by 1 and the character string C$NCONS is returned. The effect of this routine is to avoid creating new names for constants that are equal to already-named constants.

## VI. CONCLUSIONS

The derivative-taker was tested with a number of formulas and was then incorporated into the special-purpose language for least-squares fits. A sample formula and its derivatives are shown in Appendix E. The program finds these three derivatives in a minute and a half, and uses about half of the available fast core. Smaller formulas take less time and less core. Thus in the context of the language, the program provides a saving in user time and effort for a relatively small additional cost in machine time.

The derivative-taker is available only as a part of the language. Because IBM's FORMAC, which has a similar facility, is now available at Argonne, there are no plans for making this derivative-taker available as an independent program.

# APPENDIX A

## Special-purpose Language for Least-squares Fits

### Introduction

The purpose of the language is to allow a user who is unfamiliar with computers to make least-squares fits to a number of different functions. The user must define his symbols, give the function to be fitted, and list data points. The translator for the language will define an appropriate FORTRAN subprogram to be executed as part of a standard fitting procedure, Davidon's variable metric minimization program.

### Description of the Input

1. General Principles

Input is on cards or card images. Unless otherwise stated, input may appear anywhere on a card.

The equals sign (=) is used as a keyword. Thus, it should not be used except where specifically required.

Names may not begin with the letters I, J, K, L, M, or N, or end with the dollar sign ($).

The word END, appearing in columns 1-3, indicates the end of a section of data or of a case.

2. Minimal Input

2.1. Function Definition

The translator uses the following cards to generate a FORTRAN subroutine FCN$:

2.1.1. A title identifying the function. This title must be the first card and is printed every time the program is run.

The following cards may be in any order, but all must appear:

2.1.2. The symbol PARAMETERS=, followed by a list of parameter names, each having at most six characters. The names must be separated by commas. The list may occupy more than one card, but no name should appear more than once in the list.

2.1.3. The symbol IV=, followed by a list of the names of the independent variables. These names may have at most six characters each and must be separated by commas. The list may appear on more than one card, but no name should appear more than once in the list.

2.1.4. The symbol DV=, followed by the name of the dependent variable.

2.1.5. The symbol DATA=, followed by a list of independent- and dependent-variable names in the order in which they appear in the data to be fitted. If weights are given, the word WEIGHT appears in the list. If error estimates are given, the word ERROR appears in the list.

2.1.6. The explicit formula for the dependent variable. If DVNAME represents the name of the dependent variable, this appears as DVNAME= any valid FORTRAN formula. It may occupy more than one card, but if it does, no continuation marks of any kind may appear. The formula may contain the names of the independent variables and parameters. If a formula for the partial derivative of this function with respect to a parameter named PNAME appeared before this formula in the input, DPNAME, denoting this derivative, may also be used.

2.1.7. For each parameter, the partial derivative of the function with respect to that parameter. If PNAME is the name of the parameter, the statement DPNAME= DERV will call the derivative-taker. If the derivative-taker is not used, this derivative appears as DPNAME= any valid FORTRAN formula, which may contain names of parameters and independent variables. The DVNAME's and the DPNAME's may also be used if the appropriate formulas have already been given. The formula may occupy more than one card, but each derivative should start on a new card. Continuation marks are not permitted.

2.1.8. An end-of-section card, containing the word END in columns 1-3 and blanks in all other columns. This card must follow the function-definition cards.

## 2.2 Required Data for Cases

2.2.1. The first card must be a title card to identify the case. It is printed with the output for each case.

2.2.2. For the first case, an estimated value and, possibly, a standard deviation for each parameter. For cases following the first, only those values that differ from the first case need appear. Each parameter estimate must appear on one card.

One of the following forms is used:

2.2.2.1. If no standard deviation is specified,

PNAME=value.

In this case, a standard deviation of 0.1 of the value is assumed.

2.2.2.2. PNAME=value; standard deviation.

2.2.2.3. PNAME=value; CONSTANT.

In this form the value is held constant for the duration of the case.

2.2.3. After the estimates for the parameters have been given, an END card (with END in columns 1-3) must appear.

2.2.4. The observed data to be fitted then follow. For each point, numbers must be in the order given in the statement DATA= (as specified in 2.1.5.). Each number must be separated from other numbers by at least one blank and may not contain embedded blanks.

2.2.5. Two END cards must follow the last data card. If more cases are desired, more sets of cards (starting with the case title) are included.

2.3. Job-control Cards Needed (to run on System/360/50/75 under MVT OS Release 17 with ASP monitor)

The cataloged procedures VMMCLG and VMMLG are listed in Appendix D.

2.3.1. To compile the FORTRAN subroutine FCN$ and run one or more cases, the following job-control cards are needed:

```
//      EXEC VMMCLG
//GO.SYSIN DD *
   Data as described above
/*
```

Punched cards containing the names of the parameters and other information are always produced in this process.

2.3.2. If the function to be fitted is complicated, the default region for the translator may not be large enough. The following job-control cards may be used to avoid this difficulty:

```
//      EXEC VMMCLG,VMMREGN=nnnK,GOREGN=nnnK
//GO.SYSIN DD *
   Data as described in 2.1 and 2.2 above
/*
```

Here "nnn" is a number not greater than 750. VMMREGN and GOREGN must have the same value.

2.3.3. To compile and run as above and to produce an object deck of the FORTRAN subroutine, the following job-control cards are needed:

```
//      EXEC VMMCLG,OPTIONS='DECK'
//GO.SYSIN DD *
   Data cards as described in 2.1 and 2.2 above
/*
```

Under Release 17, two sets of punched output are provided by such a run. The first consists of

2.3.3.1. Burst card.

2.3.3.2. Two to five cards always obtained from the special-purpose compiler when a FORTRAN program is generated.

2.3.3.3. Burst card.

The object deck contains

2.3.3.4. Burst card.

2.3.3.5. Object deck for the generated FORTRAN subroutine FCN$. This part of the punched output contains the characters FCN$ in columns 73-76 and a four-digit sequence number, starting with 0000, in columns 77-80.

2.3.3.6. Burst card.

Only the object deck is valid input to the linkage editor.

2.3.4. To execute the program using the object deck obtained in 2.3.3.5 the following cards are needed:

```
//      EXEC VMMLG
//EDT.SYSIN DD *
    Object deck from 2.3.3.5 above
/*
//GO.SYSIN DD *
    Cards obtained in 2.3.3.2.
    Case definition data given under 2.2.
```

2.3.5. Table A.I. gives a list of the available symbolic parameters in VMMCLG and VMMLG and their default values.

TABLE A.I. Symbolic Parameters and Default Values for VMMCLG and VMMLG

| Parameter | Default Value | Procedure | Meaning |
|---|---|---|---|
| VMMREGN | 350K | VMMCLG | Region size for dummy step preceding the translation step. Value must be greater than or equal to GOREGN. |
| GOREGN | 350K | VMMCLG | Region size for translation step. |
| | 260K | VMMLG | Region size for execution of variable metric minimization program. |
| REGN | 260K | VMMCLG | Region size for execution of FORTRAN compiler. |

TABLE A.I (Contd.)

| Parameter | Default Value | Procedure | Meaning |
|---|---|---|---|
| EDTREGN | 260K | VMMCLG | Region size for execution of the linkage editor. |
| | 260K | VMMLG | Same as for VMMCLG. |
| RUNREGN | 260K | VMMCLG | Region size for execution of the variable metric minimization program. |
| EDTOPTS | 'LIST,MAP' | VMMCLG | Linkage editor options. |
| | 'LIST,MAP' | VMMLG | Linkage editor options. |
| LSIZE | '(232K,100K)' | VMMCLG | Space available to the linkage editor. |
| | '(232K,100K)' | VMMLG | Same as in VMMCLG. |
| OPTIONS | None given; i.e., standard system defaults | VMMCLG | FORTRAN compiler options. |

3. Sample Case Using Minimal Input

Suppose it is desired to find the values of a and b that give the least-squares fit of y = ax + b to the following sets of data:

Case 1

| y | x | Error in y |
|---|---|---|
| -1.0 | 0.0 | 0.02 |
| -0.8 | 0.2 | 0.03 |
| -0.25 | 1.0 | 0.01 |
| 0.4 | 2.0 | 0.04 |
| 1.2 | 0.3 | 0.03 |
| 1.9 | 4.0 | 0.02 |
| 2.6 | 5.0 | 0.01 |

Case 2

| y | x | Error in y |
|---|---|---|
| -1.0 | 0.0 | 0.01 |
| -1.4 | 1.0 | 0.01 |
| -1.8 | 2.0 | 0.02 |
| -2.2 | 3.0 | 0.03 |
| -2.7 | 4.0 | 0.06 |

The entire input might look like this (each line is a card):

```
JOB card
Accounting card
//      EXEC VMMCLG
//GO.SYSIN DD *
LINEAR FIT TO SAMPLE DATA
```

```
PARAMETERS=A,B
IV=X
DV=Y
DATA=Y,X,ERROR
Y=A*X+B
DA=DERV
DB=DERV
END
CASE 1
A=.833
B=-1.0
END
   -1.0      0.0      0.02
   -0.8      0.2      0.03
   -0.25     1.0      0.01
    0.4      2.0      0.04
    1.2      0.3      0.03
    1.9      4.0      0.02
    2.6      5.0      0.01
END
END
CASE 2
A=-.4
END
   -1.0      0.0      0.01
   -1.4      1.0      0.01
   -1.8      2.0      0.02
   -2.2      3.0      0.03
   -2.7      4.0      0.06
END
END
/*
```

After an object deck for the FORTRAN subroutine has been obtained, the following input deck is used:

```
JOB card
Accounting card
//      EXEC VMMLG
//EDT.SYSIN DD *
    Object deck as described under 2.3.3.5.
```

```
/*
//GO.SYSIN DD *
    Cards obtained in 2.3.3.2.
CASE 1
A=.833
B=-1.0
END
    Data as before, ending with
END
END
/*
```

## 4. Expanded Input

The following statements may appear in the function-definition section of the input. Except for the DATA=INSERT option, which replaces the DATA=list statement, these statements appear in addition to the function-definition cards described under "2. Minimal Input."

### 4.1. DATA FORMAT Option

A user may, if he wishes, give his own FORTRAN format for reading the observed data. The statement for specifying the format is

DATA FORMAT = any valid FORTRAN format.

The format may be continued on more than one card. The equals sign is optional here; the compiler supplies enclosing parentheses for the format. This statement causes information for each data point to be read according to the given format in the order specified in the statement DATA=list. This option is extremely slow. If at all possible, the format specification should be omitted or a DATA=INSERT option used (see 4.3 below).

### 4.2 FORTRAN Inserts

A user may extend the language by placing FORTRAN statements at six "breakpoints" in the subprogram. For instance, inserts can be used to read extra data, print information about the progress of the fitting procedure, fit a function that cannot be specified in a single statement, or plot results. Also, the code may be made more efficient by defining common subexpressions in inserts so that they are calculated only once.

4.2.1. The inserts have the following form:

| Column | Description |
|---|---|
| 1 | An integer from 0 to 5 indicating the placement of the insert ("C" for comment may also be included if it immediately follows a numbered insert). |

| Column | Description |
|---|---|
| 2-80 | Valid FORTRAN statement. To avoid duplicate statement numbers, user's statement numbers should be between 100 and 999. User's names known to the program are the names of the dependent variable, independent variables, and parameters. D followed by a parameter name is the derivative of the user's function with respect to that parameter. All other names should be defined by the user. No name may end with $. |

When the card image is inserted by the translator, the integer in column 1 is replaced by a blank. The rest of the card image is not changed in any way.

4.2.2. The six inserts are as follows:

| Insert Level | Placement and Contents |
|---|---|
| 0 | These cards are placed between a DIMENSION and a DATA statement, and therefore must contain similar nonexecutable statements. |
| 1 | These cards are placed immediately after the reading of the fitting data and are executed only once for each case. Extra data may be read here. If extra data are present, they must appear immediately before the last END statement. If information about the progress of the fitting procedure is desired, the statement "NSSW1$=1" may be included as a level 1 insert. Then the sum of the squared residuals, the values of the fitting parameters, and the values of the derivatives of the sum with respect to each parameter are printed for each iteration of the Davidon procedure. This printout is explained in Refs. 2 and 4. |
| 2 | This set of statements is executed once each time FCN$ is called. It precedes the loop defining the sum of the squared residuals and its first partial derivatives. |
| 3 | These statements are internal to the loop that defines the sum of the squared residuals. Their exact placement depends partly on the order, relative to the function definition and derivative statements, in which they appear in the input. Thus, |

```
3  P=EXP (Z*5.)
DVNAME=P+2
DZ=5.*P
3  Q=5.+6.*A
```

is compiled as

```
P=EXP (Z*5.)
Y$=P+2. (for dependent variable definition)
GGG$(1)=5.*P (if Z is the first parameter)
Q=5.+6.*A
```

| Insert Level | Placement and Contents |
|---|---|
| 3 (Contd.) | In any case, however, cards for Insert 3 appear before the sum of the squared residuals and the derivatives of the sum are defined. The statements DVNAME= and DPNAME= may be omitted from the function definition if the value of the function and its partial derivatives are given with Insert 3 cards.<br><br>Inserts are not analyzed by the derivative-taker. Therefore, if it is called, they should be used cautiously. In the example above, for instance, the statement "DZ=DERV" would give the erroneous result "GGG$(1)=0.0." |
| 4 | These inserts appear after all calculations and printing for a case are completed. The statements are executed once for each case.<br><br>If results are to be plotted, calls to plotting subroutines appear as level 4 inserts. |
| 5 | These statements are the last to appear in the generated program (except for RETURN and END and also some formats). They are executed once after all the cases have been completed. If plotting routines have been called, the buffers are cleared with an insert at level 5. |

4.2.3. Inserts of a particular level appear in the code in the same order they appeared in the input stream. For example, suppose the following cards appear in the input:

```
1        LEVEL 1, CARD 1
2        LEVEL 2, CARD 1
1        LEVEL 1, CARD 2
3        LEVEL 3, CARD 1
3        LEVEL 3, CARD 2
2        LEVEL 2, CARD 2
```

These appear as follows:

*Block of code preceding place for Insert 1*

LEVEL 1, CARD 1

LEVEL 1, CARD 2

*Code between place for Insert 1 and place for Insert 2*

LEVEL 2, CARD 1

LEVEL 2, CARD 2

*Code between place for Insert 2 and place for Insert 3*

LEVEL 3, CARD 1

LEVEL 3, CARD 2

Rest of code

Insert cards may appear anywhere in the function definition section after the title and before the END card.

4.2.4. Because of the way the job steps are organized and the data processed, the stepname of the job step in which the Davidon procedure is actually executed is different in the procedures VMMCLG and VMMLG. Thus, DD cards that refer to extra data sets used on these inserts will have DDnames qualified by different stepnames in the two procedures. For example, suppose plotting is desired. The deck structures, including a DD statement for a Calcomp tape, are as follows.

4.2.4.1. To generate a FORTRAN code and execute:

```
/*SETUP DDNAME=PLOTTAPE,DEVICE=2400-7,ID=(,,SAVE,NL)
//      EXEC VMMCLG
//GO.SYSIN DD *
    Data to generate FORTRAN code and execute
/*
//RUN.PLOTTAPE DD DSNAME=PLOT780,DISP=(NEW,KEEP),  C
//      UNIT=(2400-2,,DEFER),LABEL=(,NL)
/*
```

4.2.4.2. To execute, using an object deck for the generated FORTRAN code:

```
/*SETUP DDNAME=PLOTTAPE,DEVICE=2400-7,ID=(,,SAVE,NL)
//      EXEC VMMLG
//EDT.SYSIN DD *
    Object deck
/*
//GO.PLOTTAPE DD DSNAME=PLOT780,DISP=(NEW,KEEP),  C
//      UNIT=(2400-2,,DEFER),LABEL=(,NL)
//GO.SYSIN DD *
    Data as appropriate for executing the program using an object deck
/*
```

Thus, for a compilation the stepname is RUN, and cards referring to it follow the data. For execution from an object module, the stepname is GO, and cards referring to it immediately precede the data.

## 4.3 DATA=INSERT Option

A programmer may use Insert 1 cards to write his own READ statements for the observed data by replacing the statement DATA=list with the statement DATA=INSERT. DATA=INSERT causes the translator to omit the usual

FORTRAN statements for reading the observed data. The parameter estimates and case title are, however, read as usual. The number of data points is set to 1000 and read by the statement "READ(IN$,1002)ND$." Thus if the programmer wants to read the number of data points, he must include an appropriate READ statement in the insert. Some variable names that may be useful in programming the appropriate READ statements are

| | |
|---|---|
| IN$ | Input "tape number"--set to 5. |
| ND$ | Number of data points--set to 1000. |
| YD$(I) | Value of the dependent variable for the $I^{th}$ data point. This array is dimensioned by 1000. |
| XD$(J,I) | Value of the $J^{th}$ independent variable for the $I^{th}$ data point. J is determined by the variable's position in the list following IV=. This array is dimensioned XD$(20,1000). |
| W$(I) | Weight of the $I^{th}$ data point. This array is dimensioned W$(1000). |

When the DATA=INSERT option is used, only one END card follows the data for each case.

# APPENDIX B

## Flow Charts for the Derivative-taker

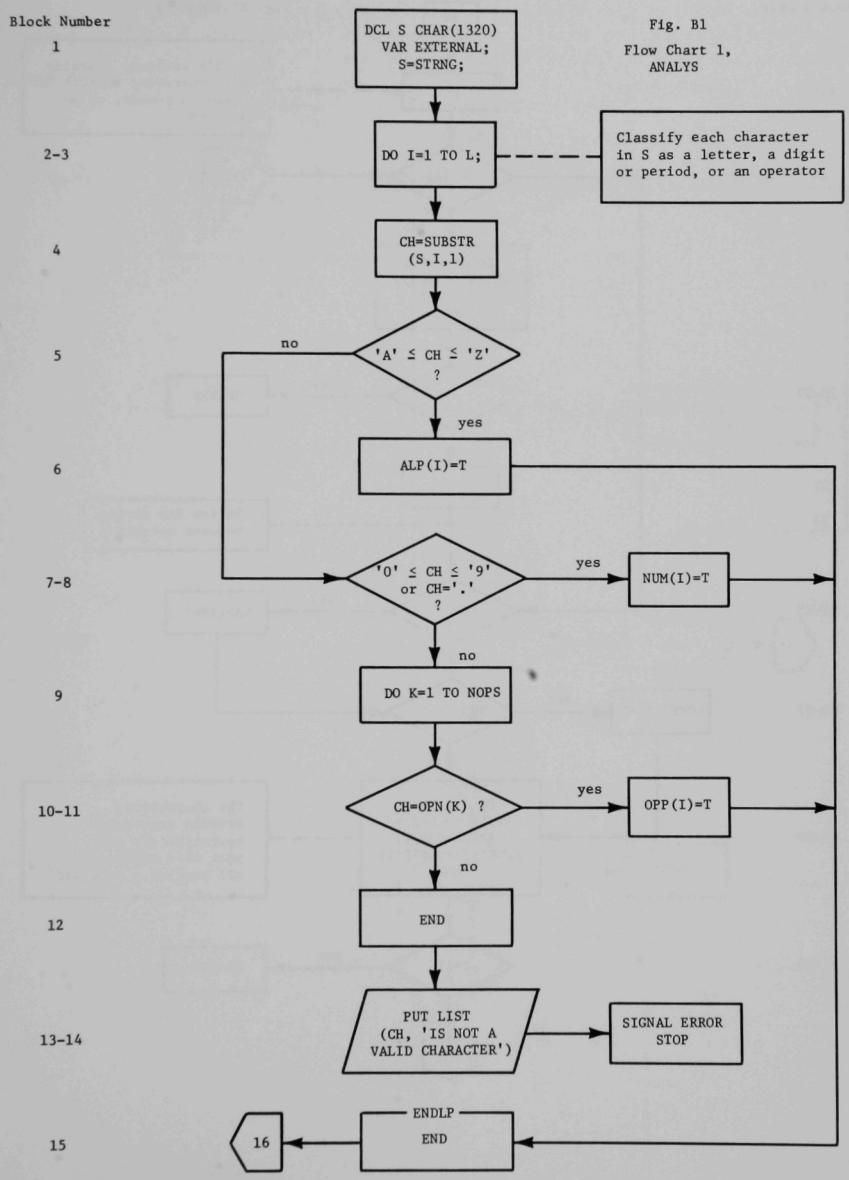



Fig. B1

Flow Chart 1, ANALYS

Fig. B1 (Contd.)

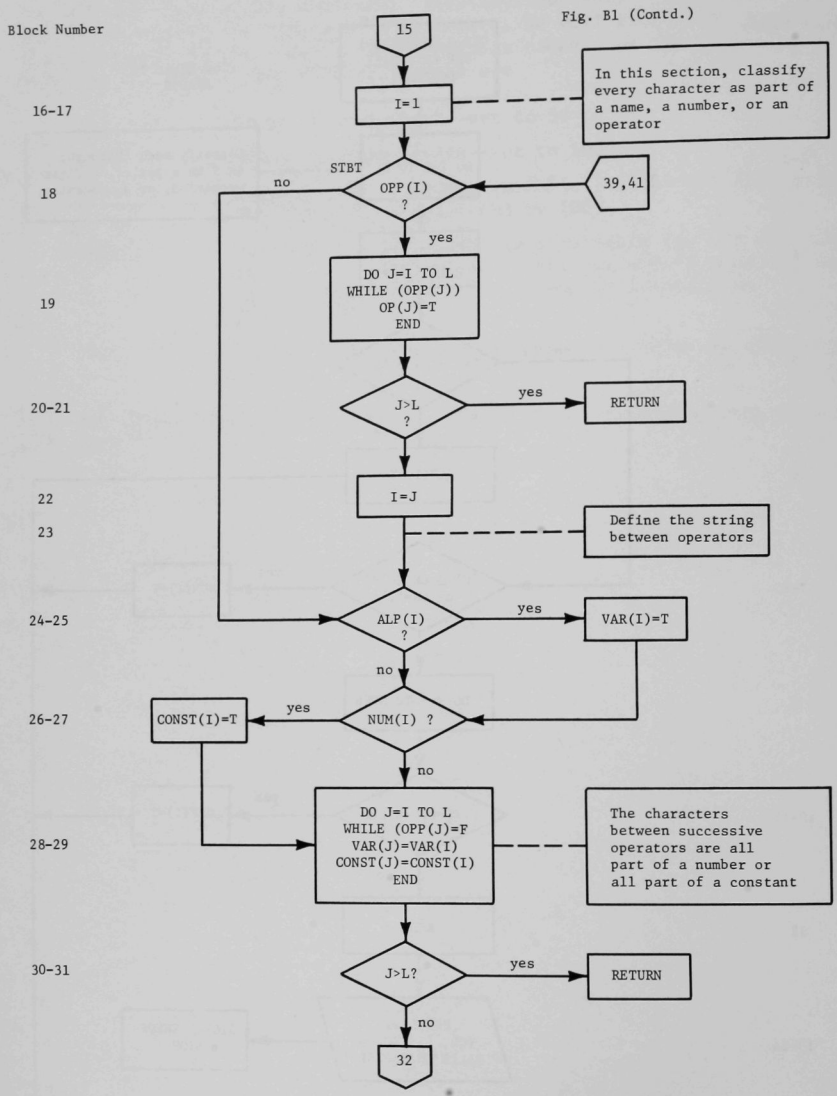

Block Number
15
16-17
I=1
In this section, classify every character as part of a name, a number, or an operator
STBT
18
OPP(I) ?
no
39,41
yes
19
DO J=I TO L
WHILE (OPP(J))
OP(J)=T
END
20-21
J>L ?
yes
RETURN
22
I=J
23
Define the string between operators
24-25
ALP(I) ?
yes
VAR(I)=T
no
26-27
NUM(I) ?
yes
CONST(I)=T
no
28-29
DO J=I TO L
WHILE (OPP(J)=F
VAR(J)=VAR(I)
CONST(J)=CONST(I)
END
The characters between successive operators are all part of a number or all part of a constant
30-31
J>L?
yes
RETURN
no
32

Fig. B1 (Contd.)

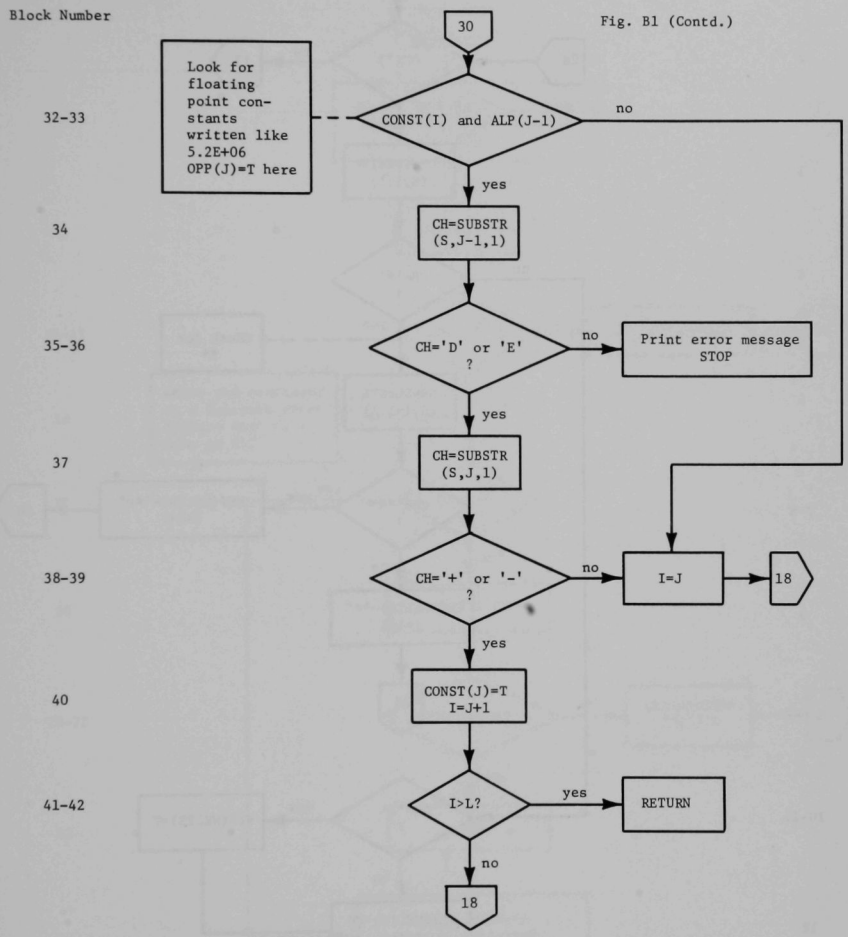

Block Number
30
Look for floating point constants written like 5.2E+06 OPP(J)=T here
32-33
CONST(I) and ALP(J-1)
no
yes
34
CH=SUBSTR (S,J-1,1)
35-36
CH='D' or 'E' ?
no
Print error message STOP
yes
37
CH=SUBSTR (S,J,1)
38-39
CH='+' or '-' ?
no
I=J
18
yes
40
CONST(J)=T
I=J+1
41-42
I>L?
yes
RETURN
no
18

Fig. B2

Flow Chart 2, GTNAM

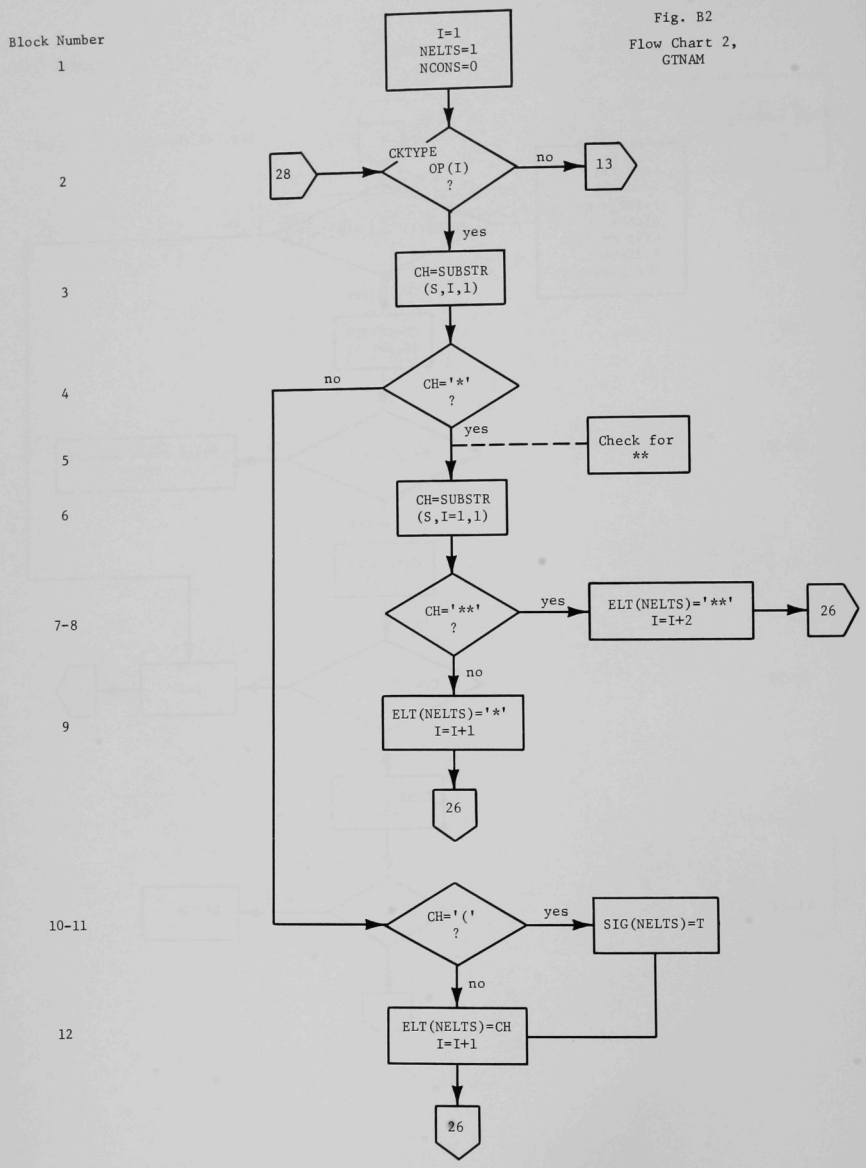


Fig. B2 (Contd.)

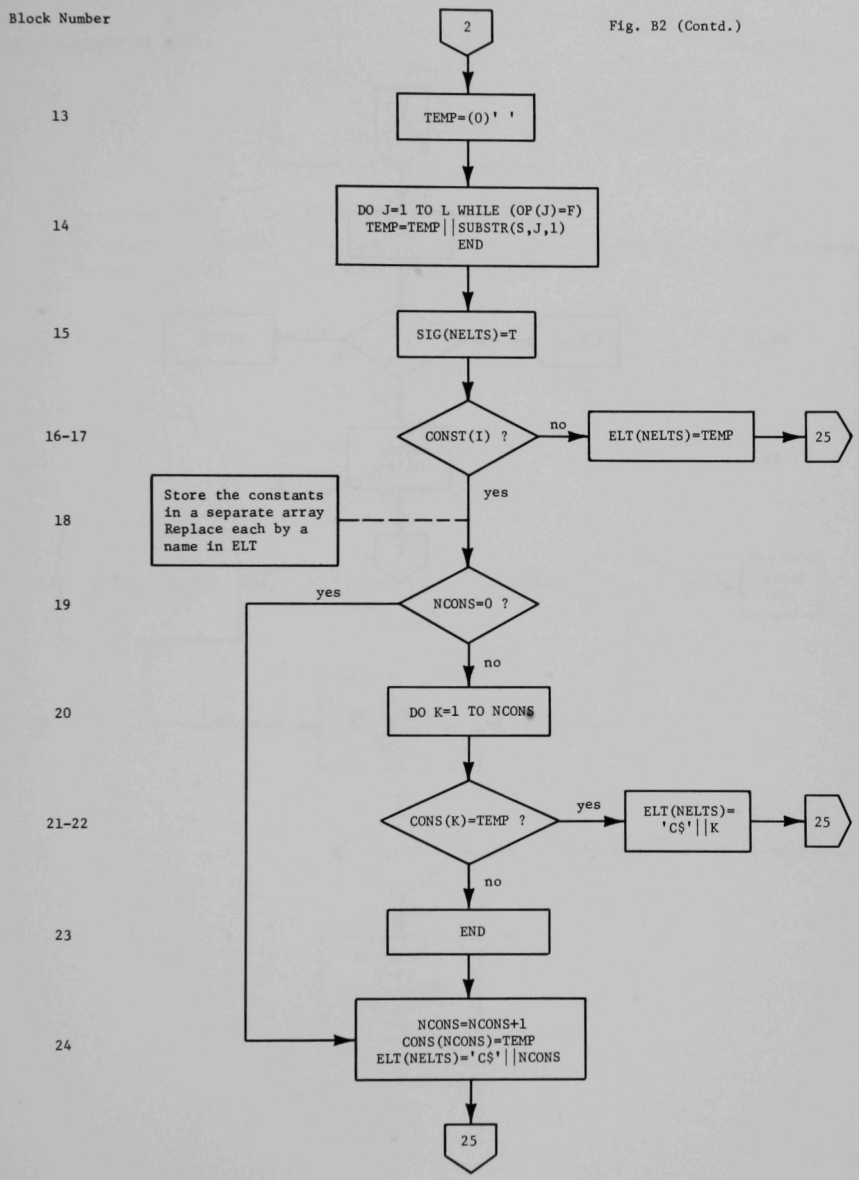

Block Number
2
13
TEMP=(0)' '
14
DO J=1 TO L WHILE (OP(J)=F)
TEMP=TEMP||SUBSTR(S,J,1)
END
15
SIG(NELTS)=T
16-17
CONST(I) ?
no
ELT(NELTS)=TEMP
25
yes
18
Store the constants in a separate array Replace each by a name in ELT
19
NCONS=0 ?
yes
no
20
DO K=1 TO NCONS
21-22
CONS(K)=TEMP ?
yes
ELT(NELTS)= 'C$'||K
25
no
23
END
24
NCONS=NCONS+1
CONS(NCONS)=TEMP
ELT(NELTS)='C$'||NCONS
25

Fig. B2 (Contd.)

Block Number

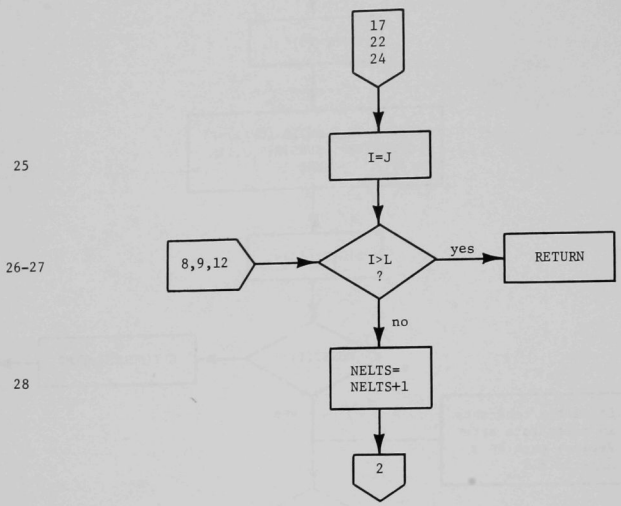


Fig. B3

Flow Chart 3, BACKUS

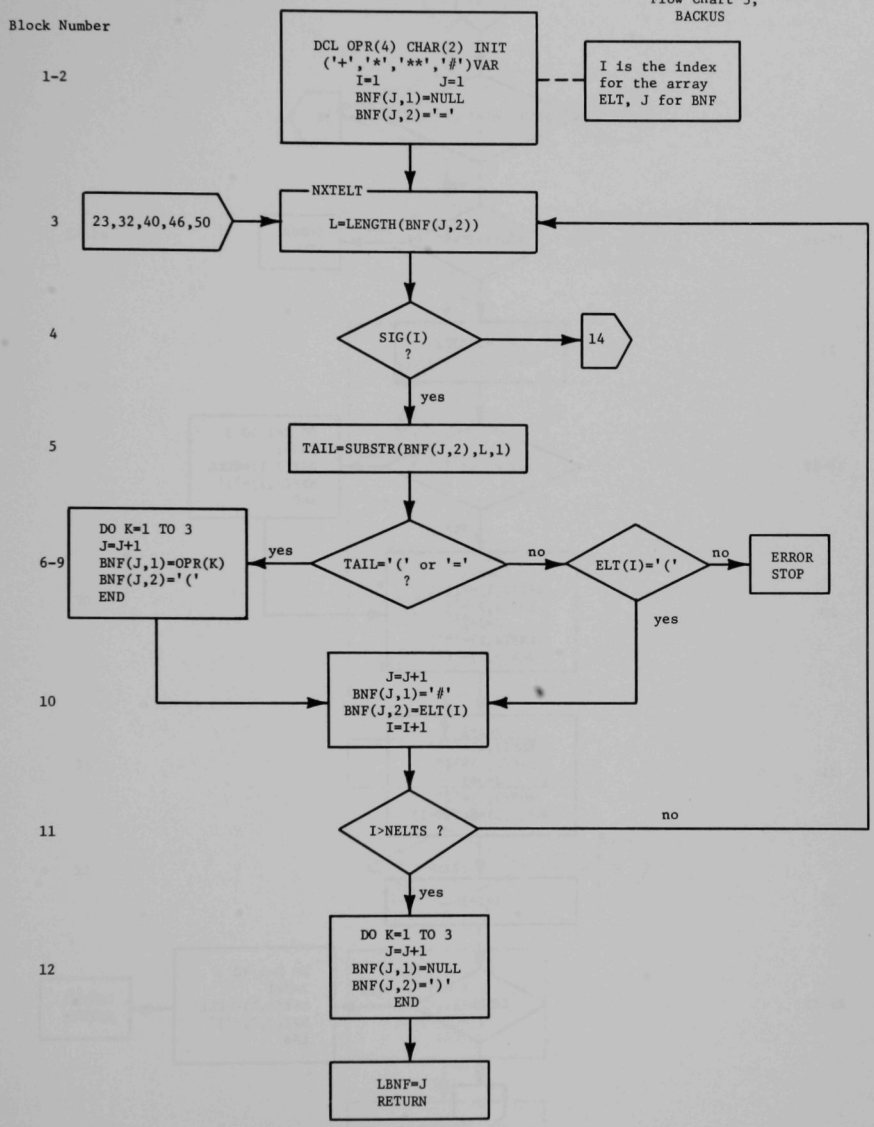
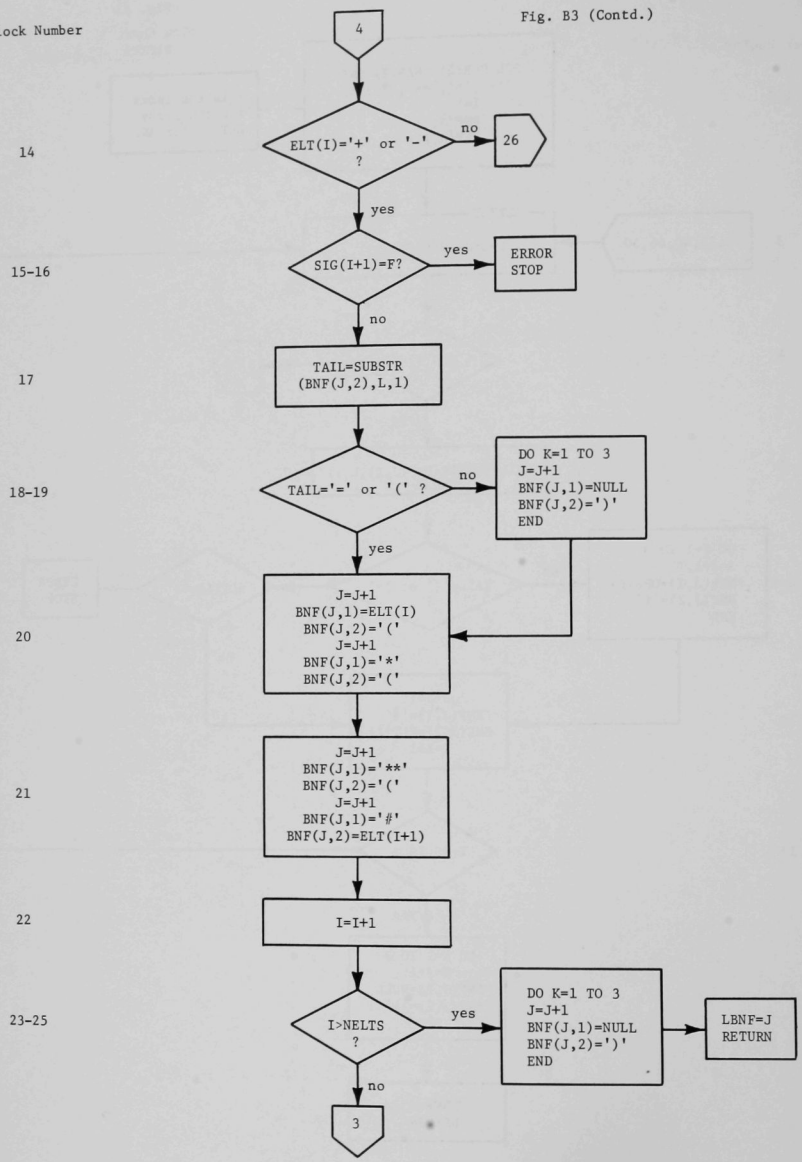


Fig. B3 (Contd.)


Block Number
4
14
ELT(I)='+' or '-' ?
no
26
yes
15-16
SIG(I+1)=F?
yes
ERROR STOP
no
17
TAIL=SUBSTR (BNF(J,2),L,1)
18-19
TAIL='=' or '(' ?
no
DO K=1 TO 3
J=J+1
BNF(J,1)=NULL
BNF(J,2)=')'
END
yes
20
J=J+1
BNF(J,1)=ELT(I)
BNF(J,2)='('
J=J+1
BNF(J,1)='*'
BNF(J,2)='('
21
J=J+1
BNF(J,1)='**'
BNF(J,2)='('
J=J+1
BNF(J,1)='#'
BNF(J,2)=ELT(I+1)
22
I=I+1
23-25
I>NELTS ?
yes
DO K=1 TO 3
J=J+1
BNF(J,1)=NULL
BNF(J,2)=')'
END
LBNF=J
RETURN
no
3

Block Number

Fig. B3 (Contd.)

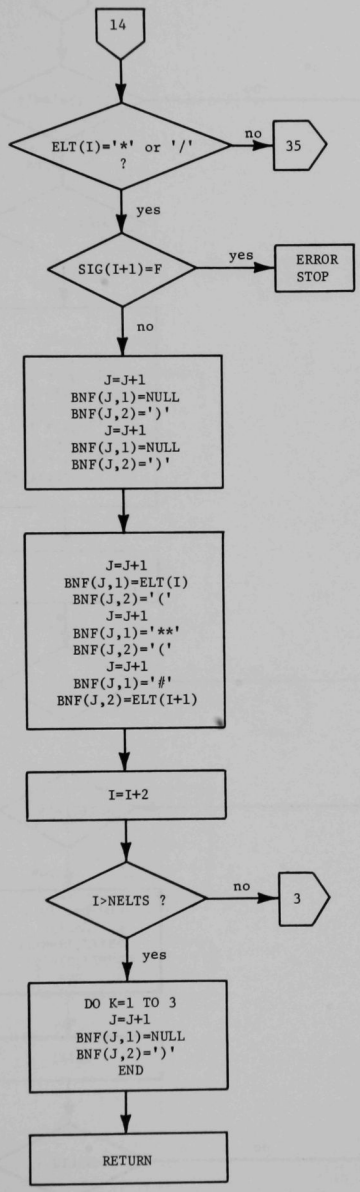


Fig. B3 (Contd.)

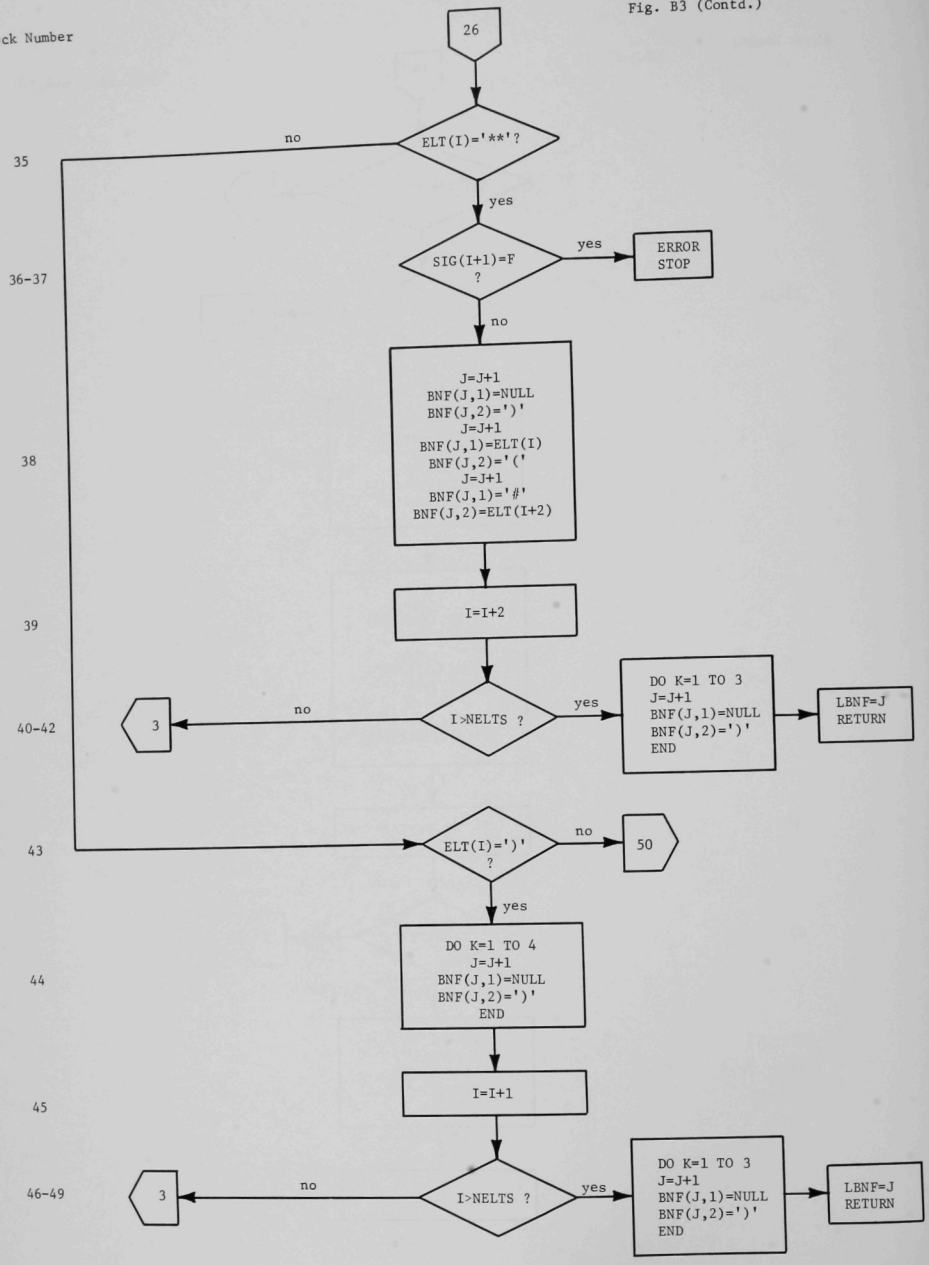

Block Number
26
35
ELT(I)='**'?
no
yes
36-37
SIG(I+1)=F ?
yes
ERROR STOP
no
38
J=J+1
BNF(J,1)=NULL
BNF(J,2)=')'
J=J+1
BNF(J,1)=ELT(I)
BNF(J,2)='('
J=J+1
BNF(J,1)='#'
BNF(J,2)=ELT(I+2)
39
I=I+2
40-42
I>NELTS ?
no
3
yes
DO K=1 TO 3
J=J+1
BNF(J,1)=NULL
BNF(J,2)=')'
END
LBNF=J
RETURN
43
ELT(I)=')' ?
no
50
yes
44
DO K=1 TO 4
J=J+1
BNF(J,1)=NULL
BNF(J,2)=')'
END
45
I=I+1
46-49
I>NELTS ?
no
3
yes
DO K=1 TO 3
J=J+1
BNF(J,1)=NULL
BNF(J,2)=')'
END
LBNF=J
RETURN

Fig. B3 (Contd.)

Block Number

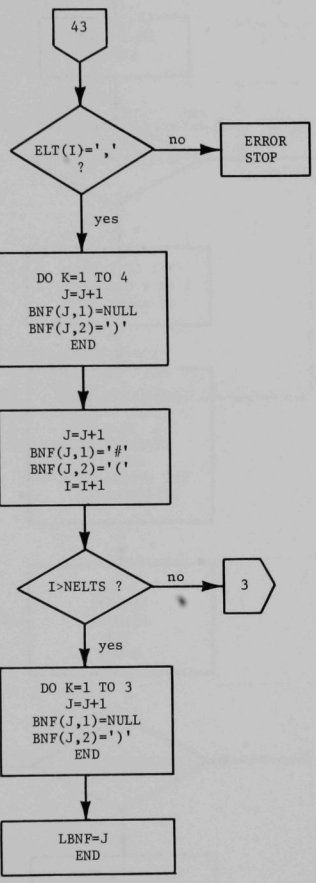


Fig. B4

Flow Chart 4,
MKTRPL

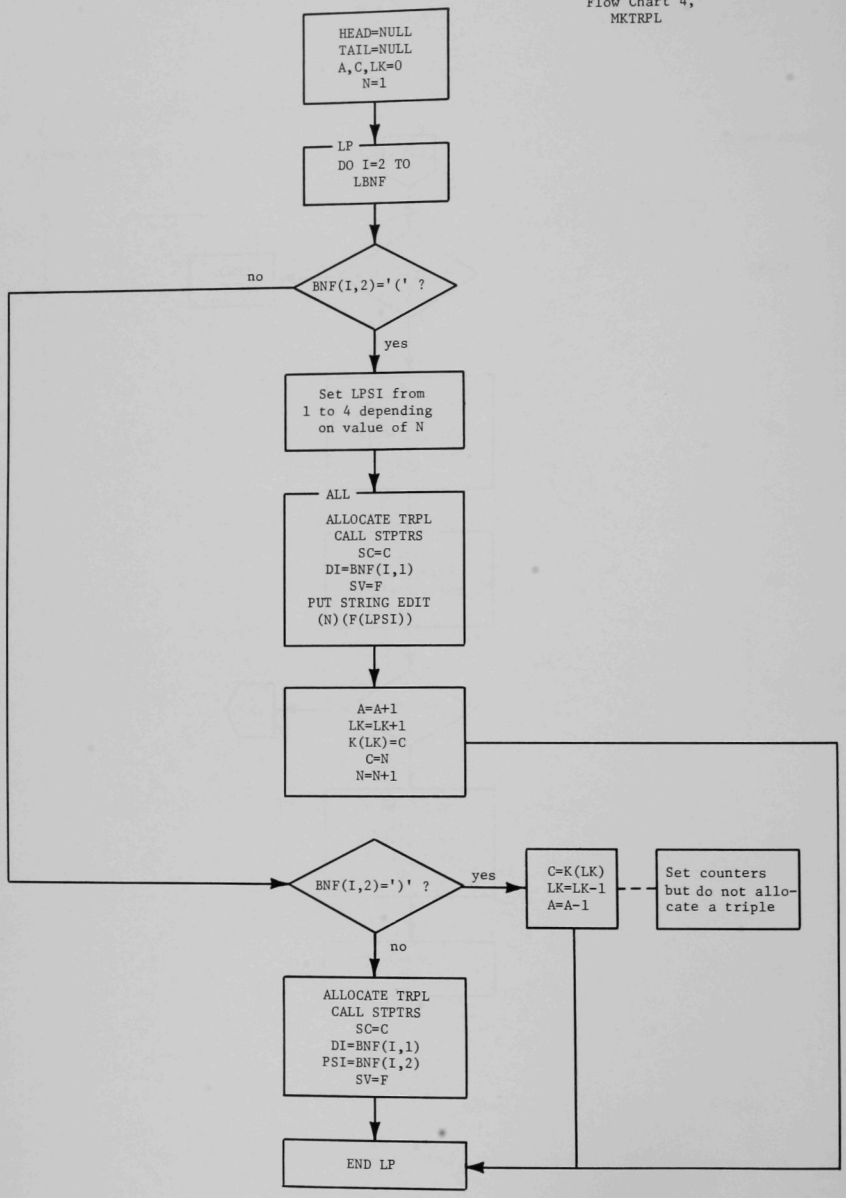
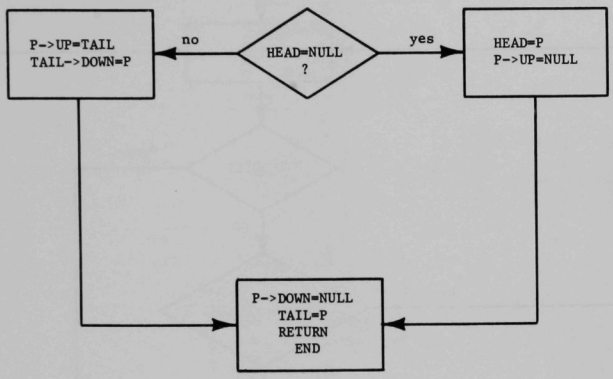


Fig. B5

Flow Chart 5,
STPTRS



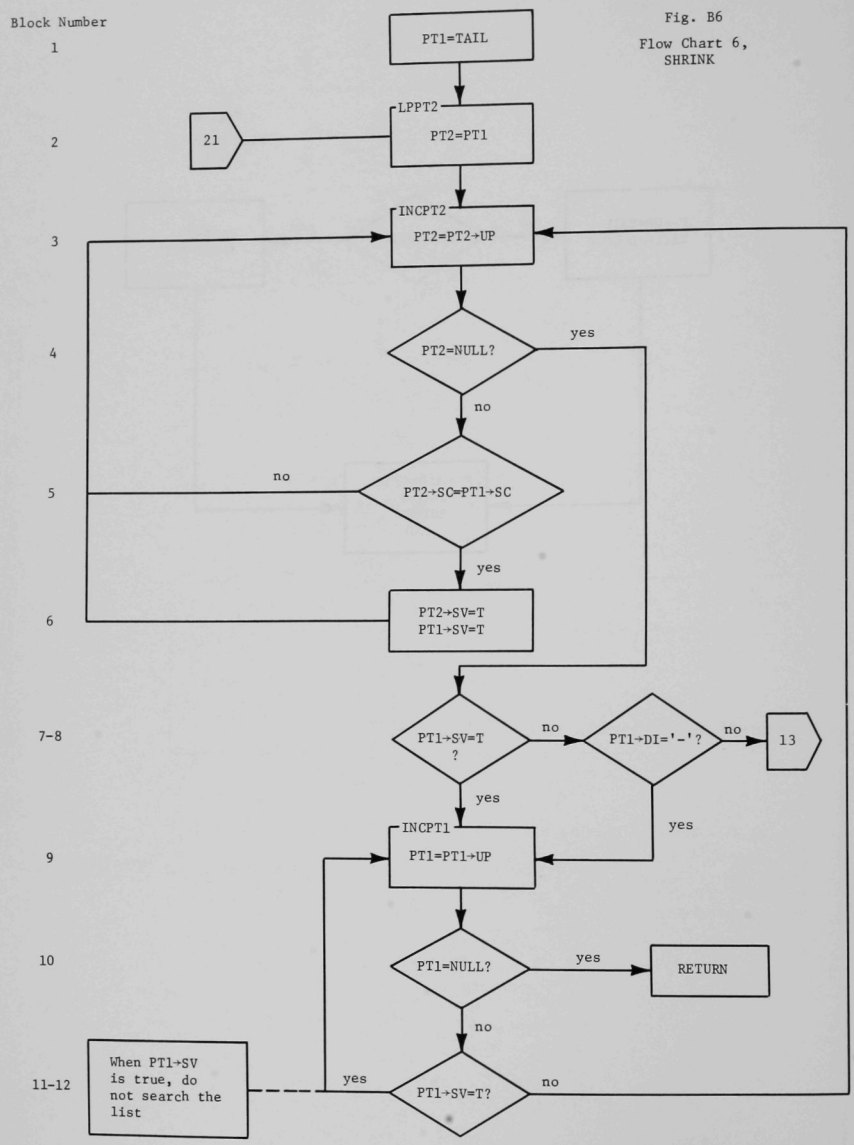
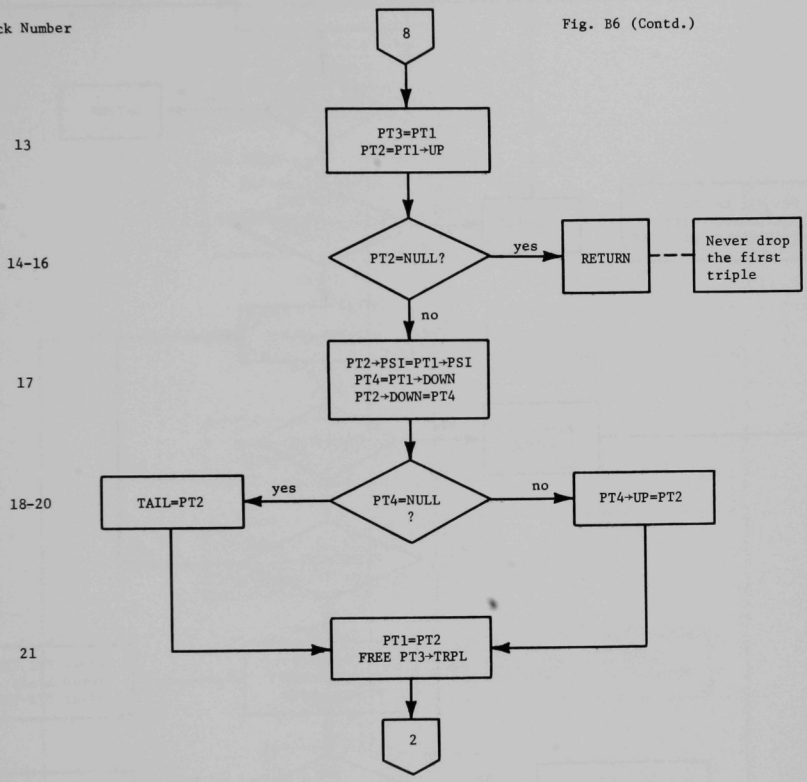



Fig. B6

Flow Chart 6, SHRINK

Fig. B6 (Contd.)


Block Number
8
13
PT3=PT1
PT2=PT1→UP
14-16
PT2=NULL?
yes
RETURN
Never drop the first triple
no
17
PT2→PSI=PT1→PSI
PT4=PT1→DOWN
PT2→DOWN=PT4
18-20
PT4=NULL ?
yes
TAIL=PT2
no
PT4→UP=PT2
21
PT1=PT2
FREE PT3→TRPL
2

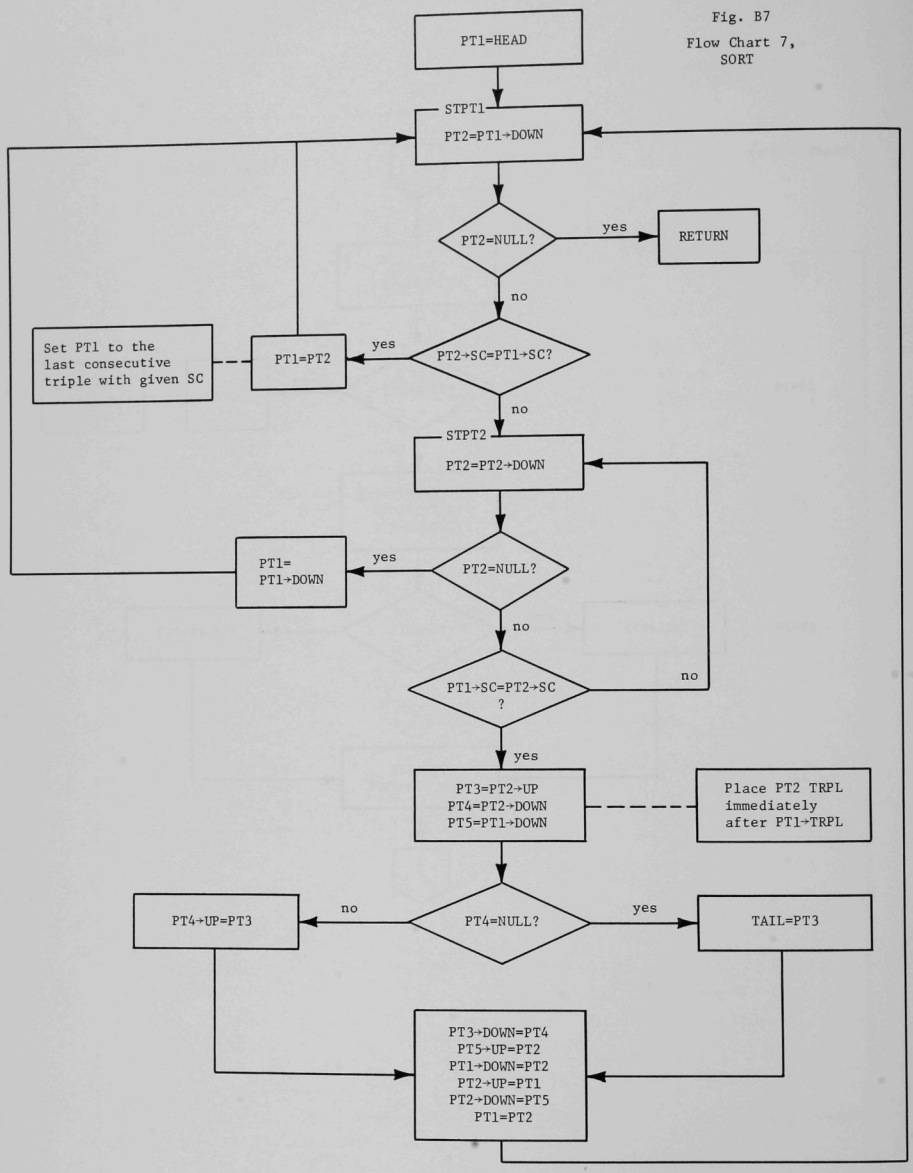



Fig. B7

Flow Chart 7, SORT

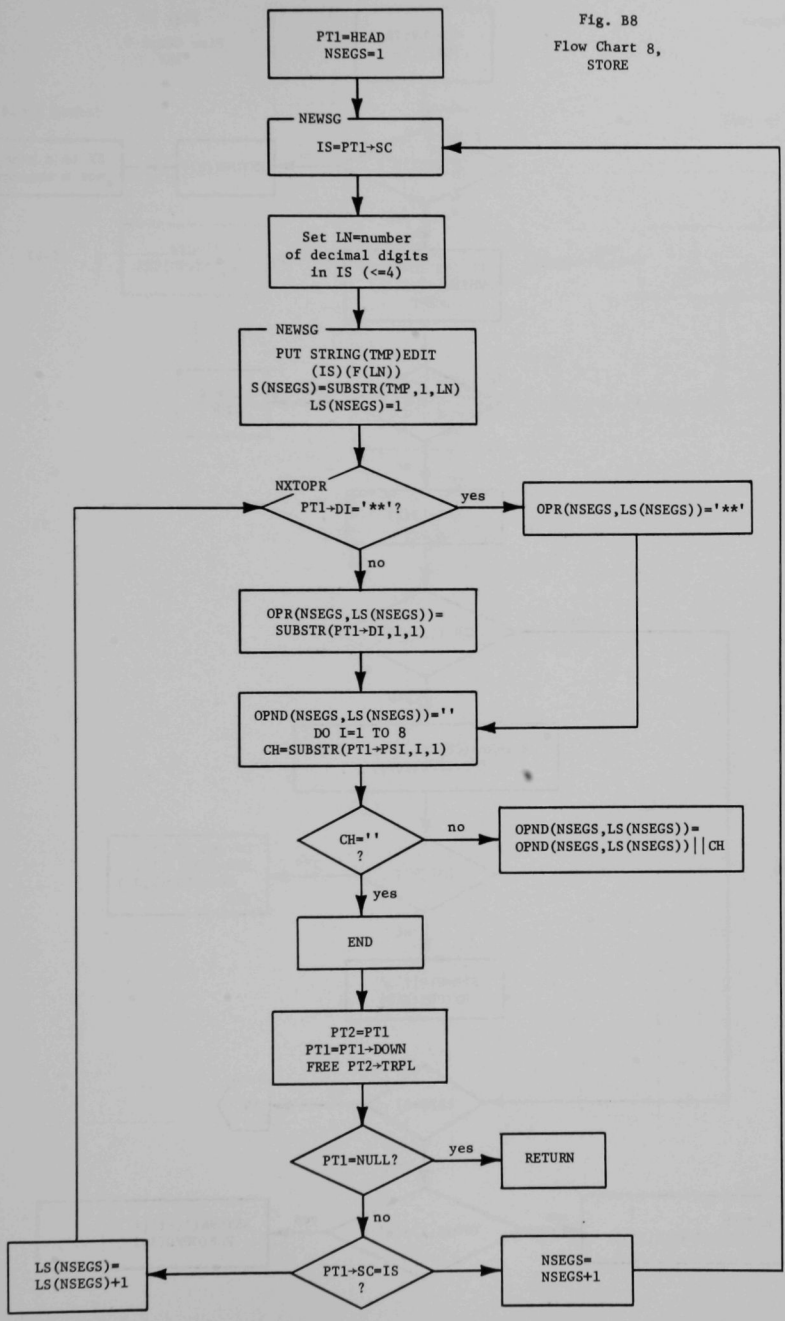
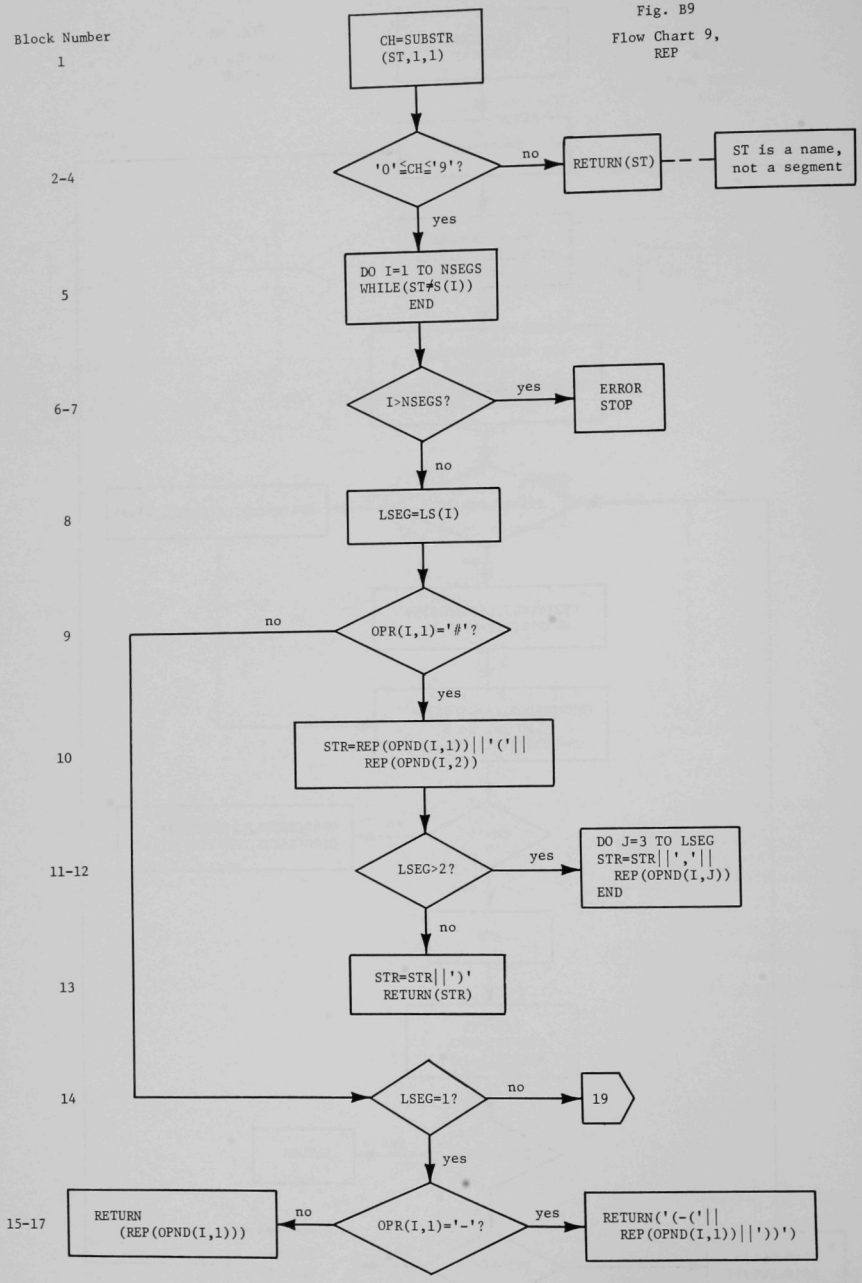



Fig. B8

Flow Chart 8,
STORE

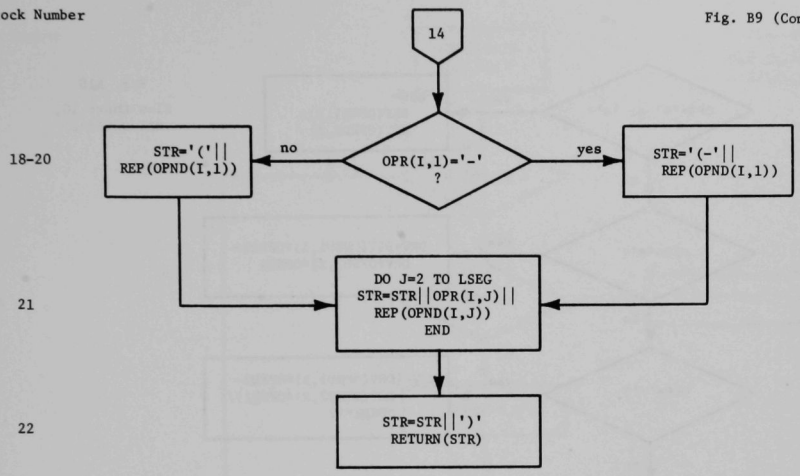
Fig. B9

Flow Chart 9, REP



Block Number

Fig. B9 (Contd.)



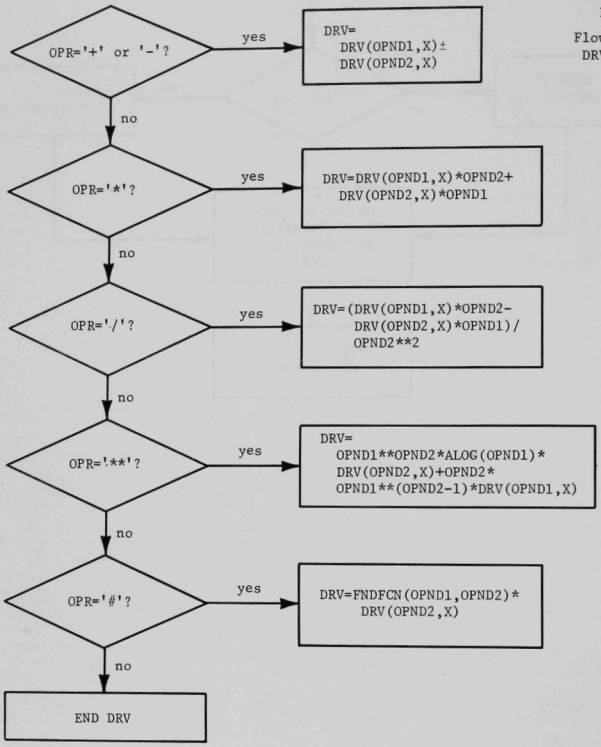



Fig. B10

Flow Chart 10,
DRV Outline

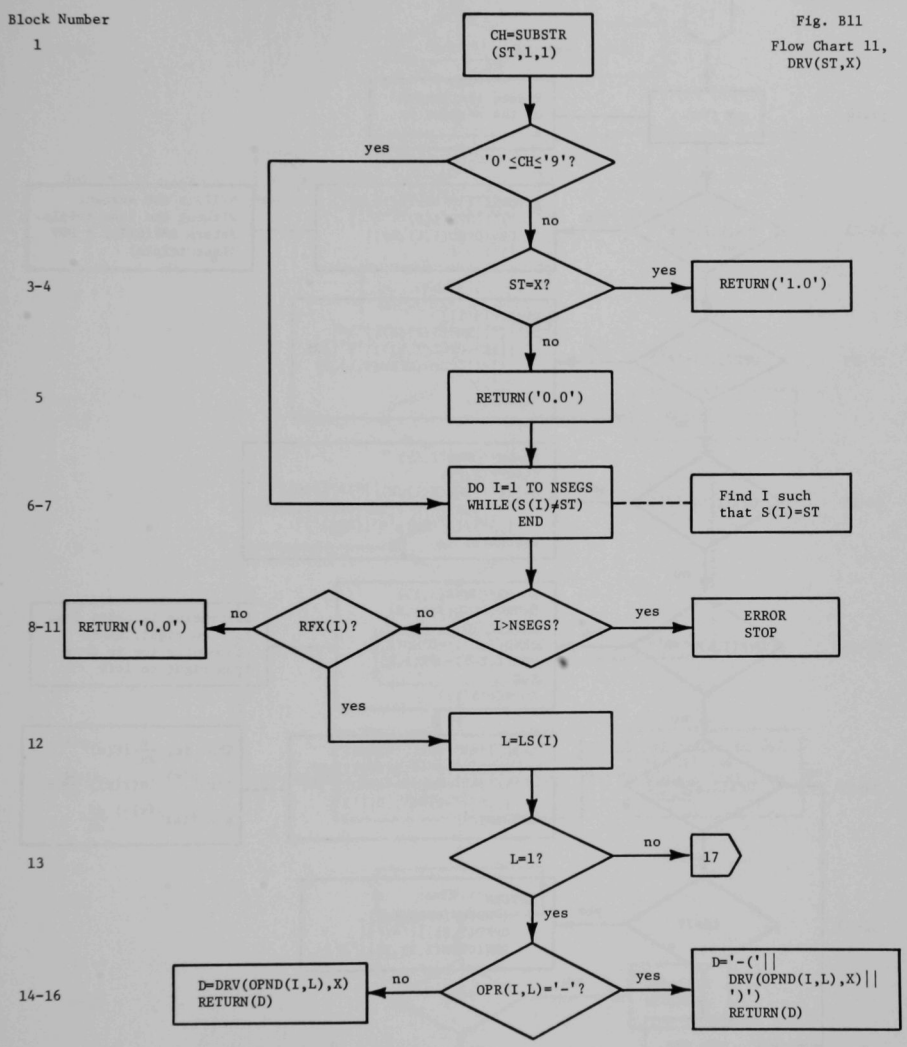



Fig. B11

Flow Chart 11, DRV(ST,X)

Fig. B11 (Contd.)

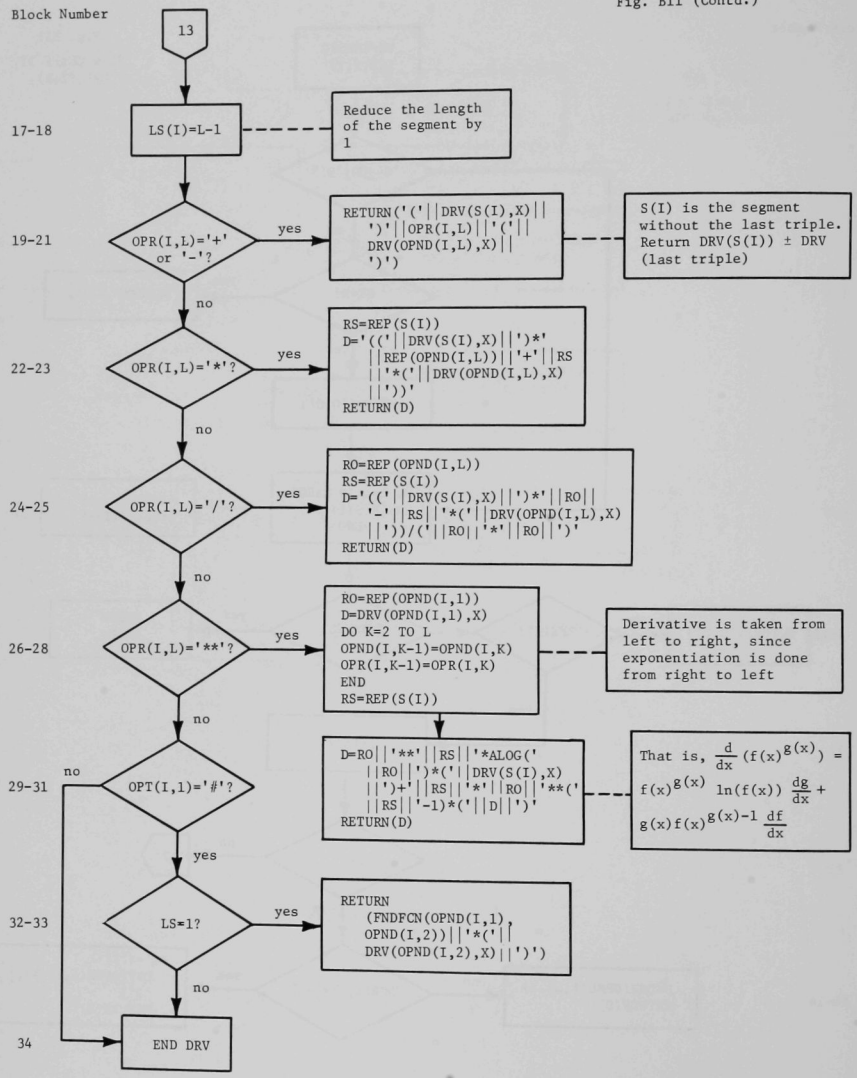


Fig. B12

Flow Chart 12, RPLCNS(STR)

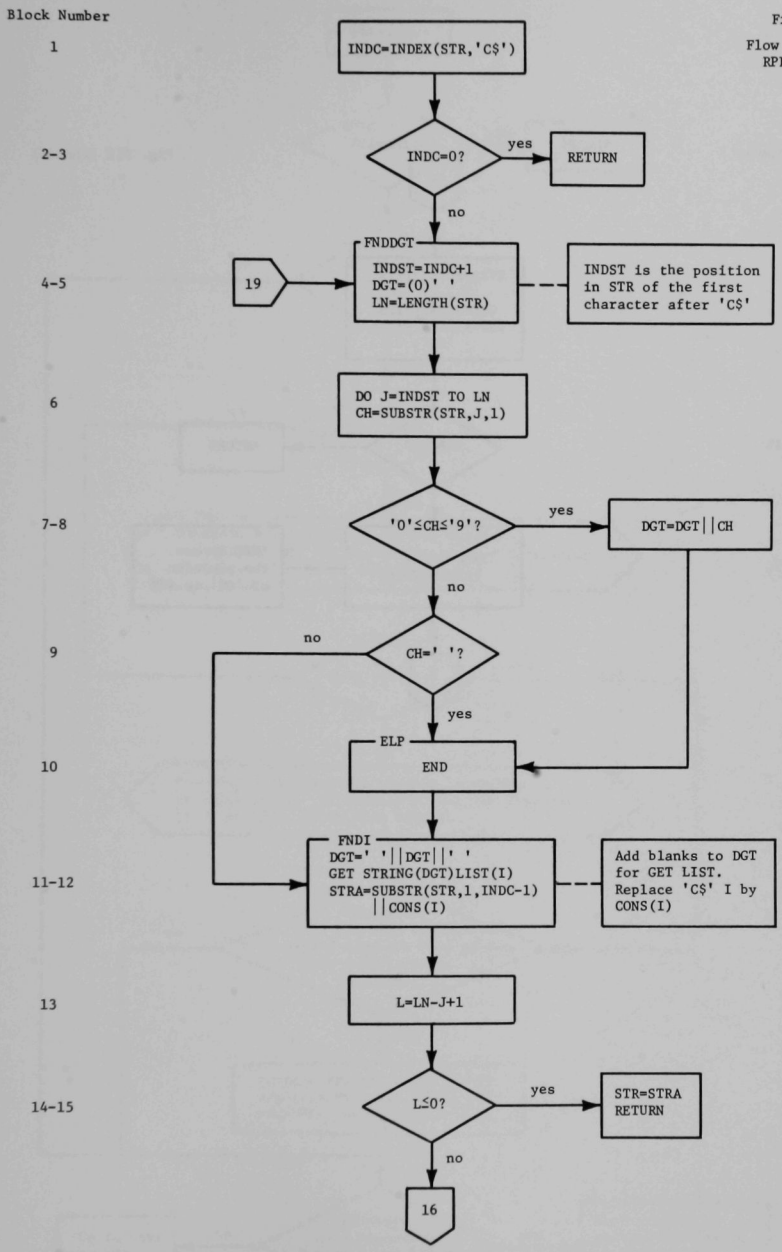


Fig. B12 (Contd.)

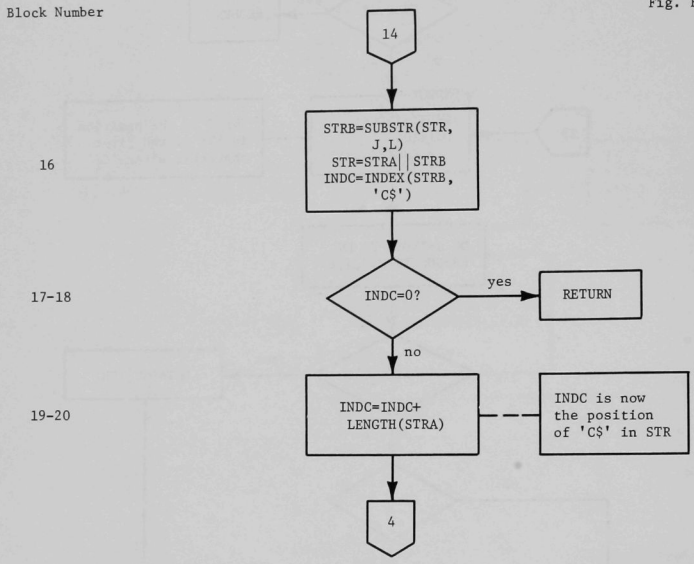


Fig. B13

Flow Chart 13, SIMP Outline

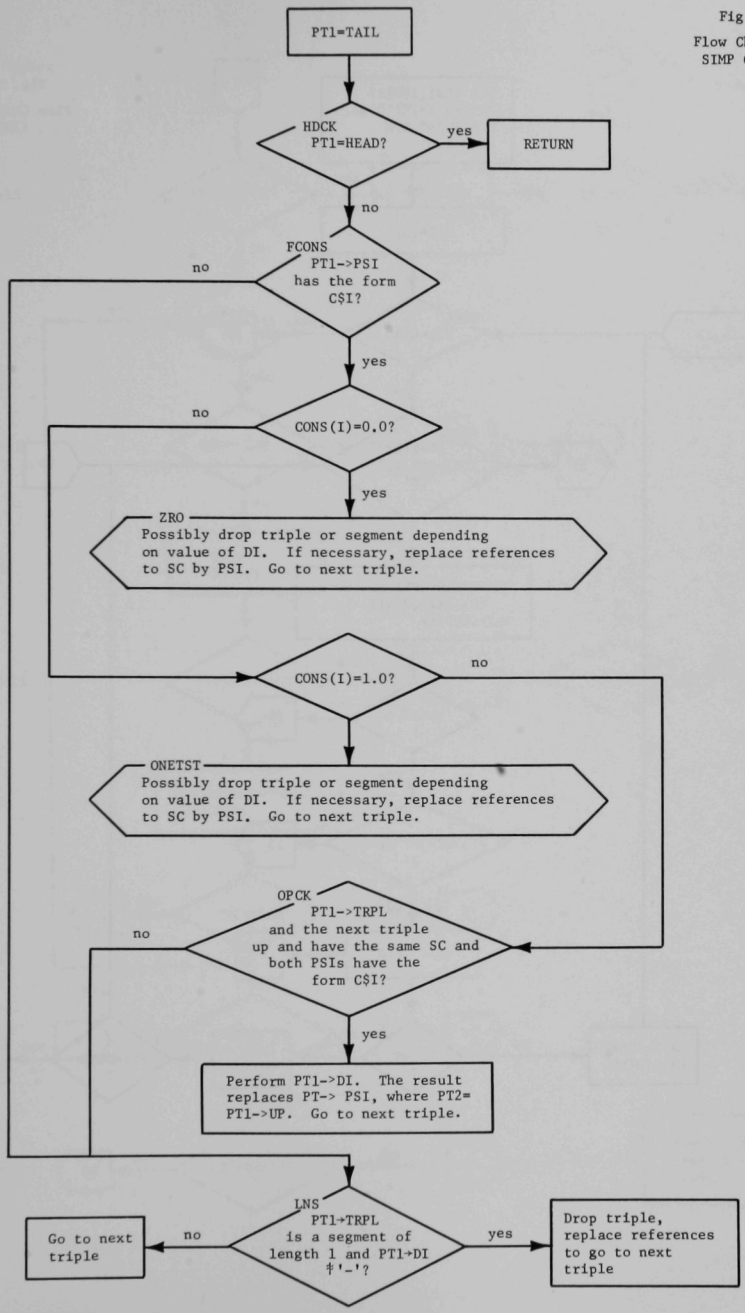
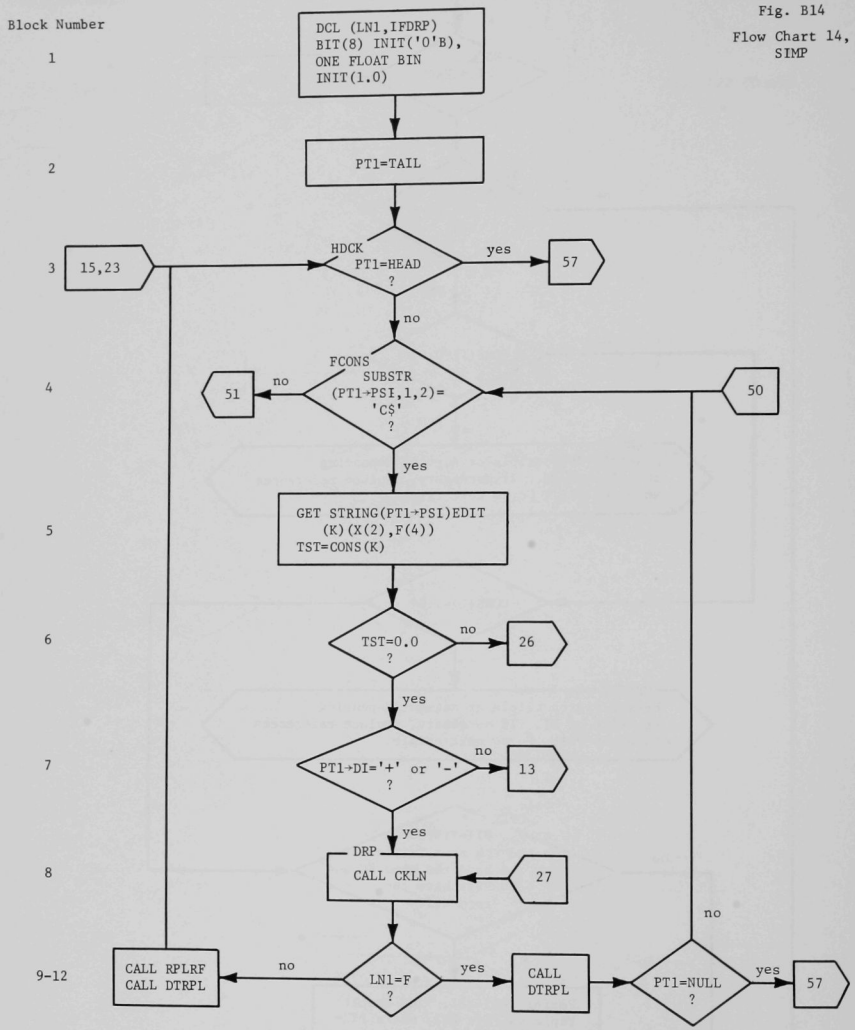


Fig. B14

Flow Chart 14, SIMP



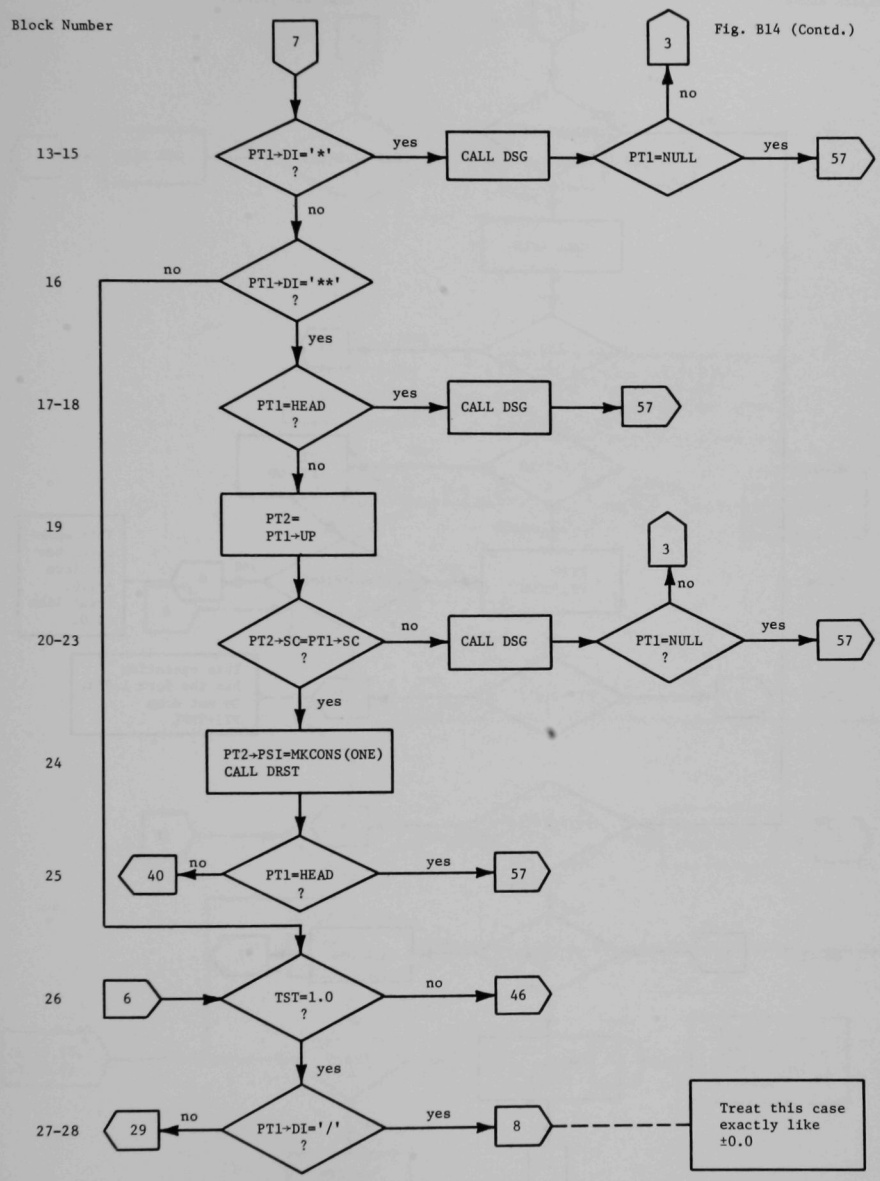
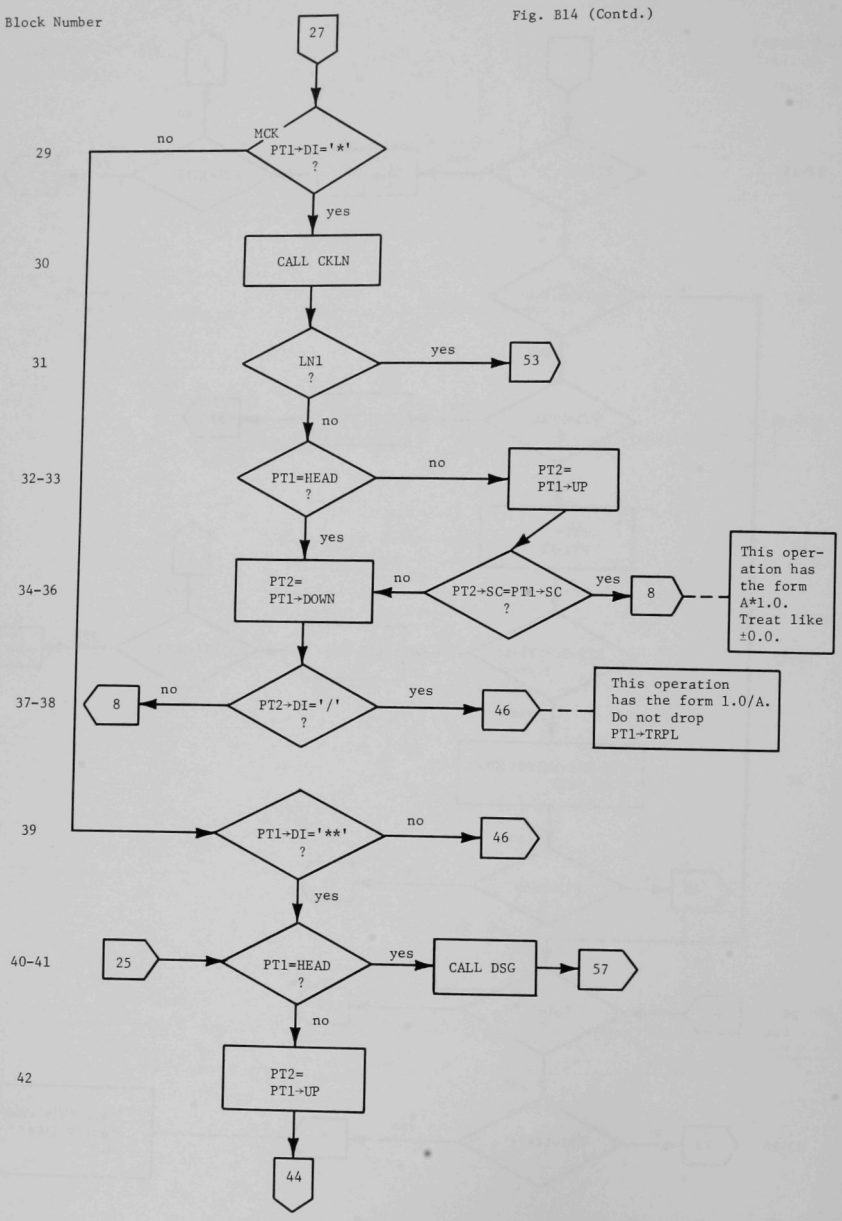



Fig. B14 (Contd.)

Fig. B14 (Contd.)



Block Number

Fig. B14 (Contd.)

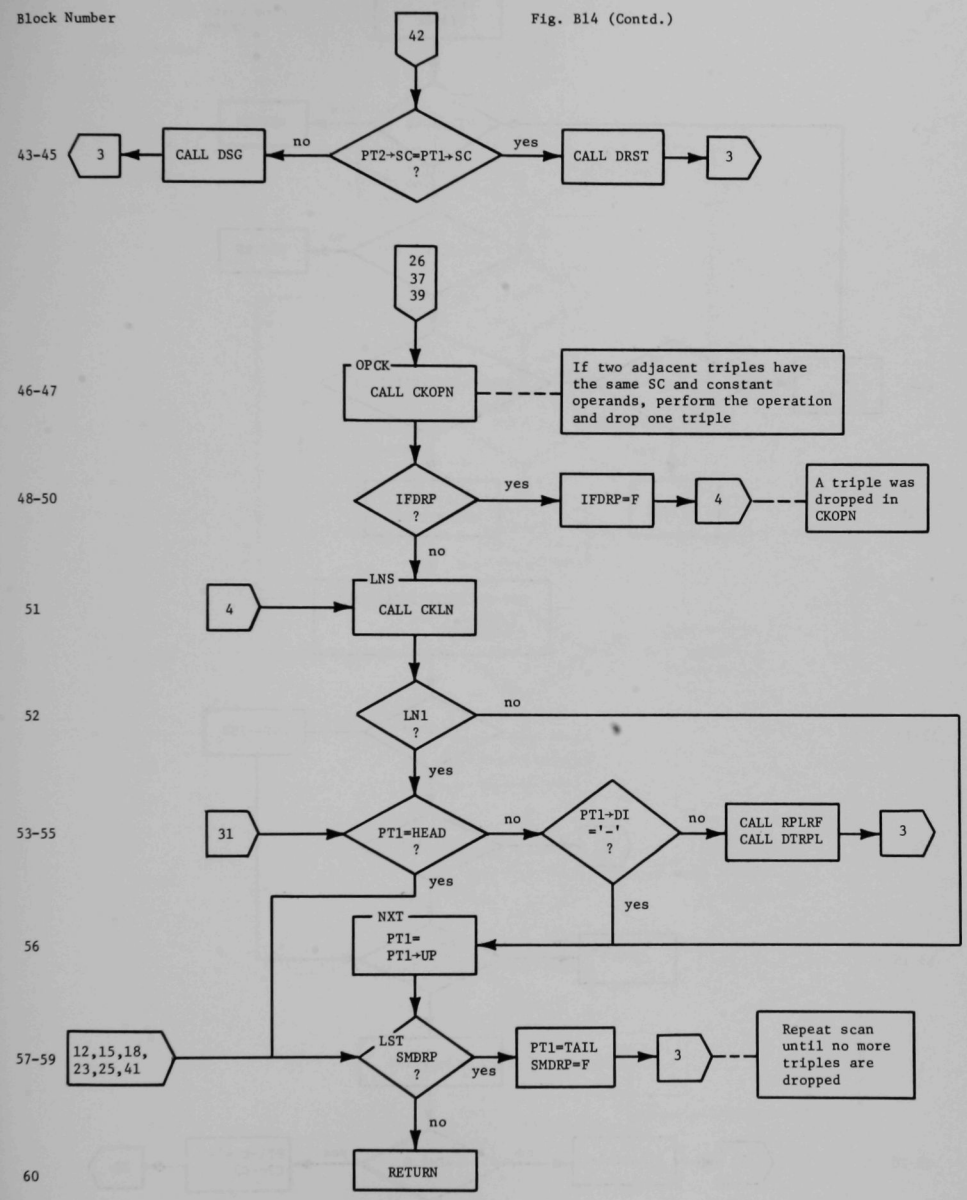
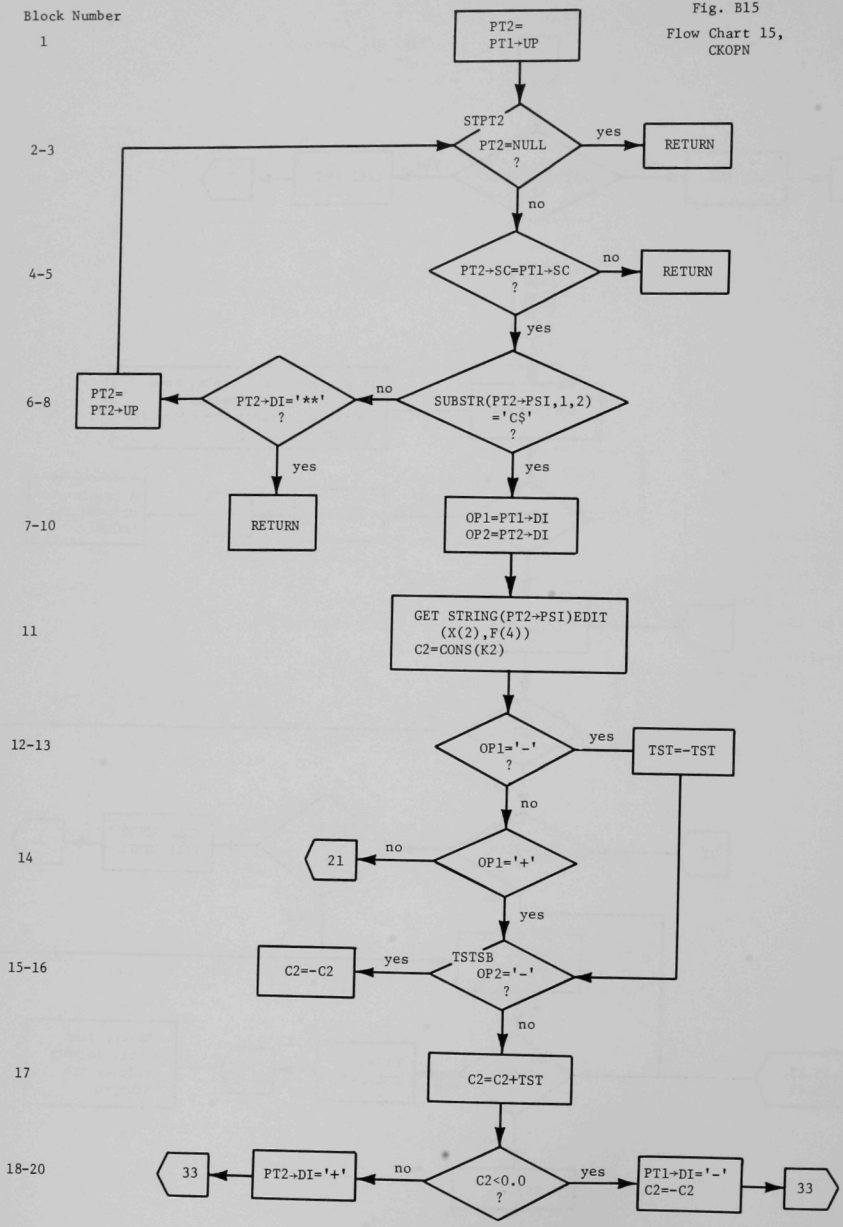


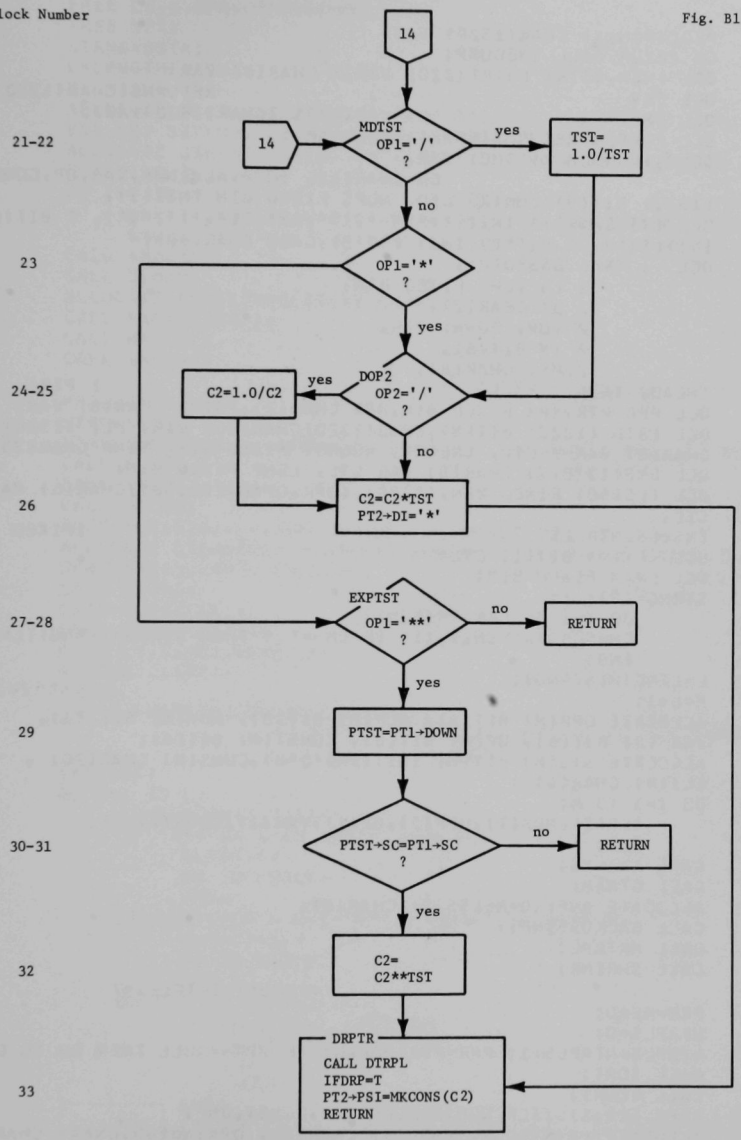
Fig. B15

Flow Chart 15, CKOPN



Block Number

Fig. B15 (Contd.)



# APPENDIX C

## Listing of DERV

```
DERV:   PROC(FCN,X) CHAR(1320) VAR;
        ON ERROR CALL IHEDUMP;
        DCL (FCN,DSTR) CHAR(1320) VAR,X CHAR(6) VAR;
                                                RETURNS(CHAR(1320)VAR);
        DCL DRV
        DCL REP                         RETURNS (CHAR(1320)VAR);
        DCL FNDFCN RETURNS(CHAR(1320) VAR);
        DCL STRNG CHAR(1320) VAR,
                                 CH CHAR(1), (OPP,ALP,NUM,VAR,OP,CONST)
        (1320) BIT(8) CONTROLLED, NOPS FIXED BIN INIT(7),
        OPN (7) CHAR(1) INIT('+','-','*','/','(',')',','), T BIT(8)
        INIT('1'B),F BIT(8) INIT ('0'B),CARD CHAR(80);
        DCL  1 TRPL BASED(P),
                 2 (SC,LP) FIXED BIN,
                 2 DI CHAR(2),
                 2 (UP, DOWN) PTR,
                 2 SV BIT(8),
                 2 PSI CHAR(8),
        (HEAD, TAIL                                          ) PTR;
        DCL PPR PTR,SPR FIXED BIN,DPR CHAR(2),PSIPR CHAR(8) VAR;
        DCL (SIG (1320) BIT(8),CONS(1320)CHAR(20) VAR, ELT (1320)
        CHAR(6) VAR ) CTL, (NELTS, NCONS) FIXED BIN, TEMP CHAR(20) VAR;
        DCL BNF(1320,2) CHAR(8) VAR CTL, LBNF FIXED BIN;
        DCL (LS(50) FIXED BIN,(S(50),(OPR,OPND)(50,50))CHAR(6) VAR)
        CTL,
        (NSEGS,NTRPLS                                        )FIXED BIN;
        DCL RFX(*) BIT(1) CTL;
        DCL LMAX FIXED BIN;
        STRNG=(0)' ';
              DO I=1 TO LENGTH(FCN);
              CH=SUBSTR(FCN,I,1); IF CH¬=' ' THEN STRNG=STRNG||CH;
              END;
        L=LENGTH(STRNG);
        M=L+1;
        ALLOCATE OPP(M) BIT(8), ALP(M) BIT(8), NUM(M) BIT (8),
        VAR (M) BIT(8), OP(M) BIT(8), CONST(M) BIT(8);
        ALLOCATE SIG(M) BIT(8) INIT((M)'0'B),CONS(M) CHAR(20) ,
        ELT(M) CHAR(6) ;
        DO I=1 TO M;
             ALP(I),NUM(I),OPP(I),OP(I),VAR(I),CONST(I)=F;
             END;
        CALL ANALYS;
        CALL GTNAM;
        ALLOCATE BNF(10*NELTS,2) CHAR(8);
        CALL BACKUS(BNF);
        CALL MKTRPL;
        CALL SHRINK;
                                        /* COUNT TRIPLES*/
        PPR=HEAD;
        NTRPLS=0;
CLP:    NTRPLS=NTRPLS+1; PPR=PPR->DOWN; IF PPR¬=NULL THEN GO TO CLP;
        CALL SORT;
        CALL COUNT;
        FREE ELT,SIG,ALP,NUM,OPP,OP,VAR,CONST,BNF;
        ALLOCATE LS(NSEGS),S(NSEGS) CHAR(6), OPR(NSEGS,LMAX) CHAR(6),
        OPND(NSEGS,LMAX) CHAR(6);
        CALL STORE;
```

```
        CALL CKRF;
        DSTR=DRV(S(1),X);
        CALL RPLCNS(DSTR);
        FREE LS,S,OPR,OPND,CONS;
        FREE RFX;
        STRNG=DSTR;
        L=LENGTH(STRNG);
        M=L+1;
        ALLOCATE OPP(M) BIT(8), ALP(M) BIT(8), NUM(M) BIT (8),
        VAR (M) BIT(8), OP(M) BIT(8), CONST(M) BIT(8);
        ALLOCATE SIG(M) BIT(8) INIT((M)'0'B),CONS(M) CHAR(20) ,
        ELT(M) CHAR(6) ;
        DO I=1 TO M;
             ALP(I),NUM(I),OPP(I),OP(I),VAR(I),CONST(I)=F;
             END;
        CALL ANALYS;
        CALL GTNAM;
        ALLOCATE BNF(10*NELTS,2) CHAR(8);
        CALL BACKUS(BNF);
        CALL MKTRPL;
        CALL SHRINK;
                                        /* COUNT TRIPLES*/
        PPR=HEAD;
        NTRPLS=0;
CLP2:   NTRPLS=NTRPLS+1; PPR=PPR->DOWN; IF PPR¬=NULL THEN GO TO CLP2;
        CALL SORT;
        CALL SIMP;
        CALL COUNT;
        FREE ELT,SIG,ALP,NUM,OPP,OP,VAR,CONST,BNF;
        ALLOCATE LS(NSEGS),S(NSEGS) CHAR(6), OPR(NSEGS,LMAX) CHAR(6) ,
        OPND(NSEGS,LMAX) CHAR(6);
        CALL STORE;
        DSTR=REP(S(1));
        CALL RPLCNS(DSTR);
        FREE LS,S,OPR,OPND,CONS;
        RETURN (DSTR);
ANALYS :PROC;
        DCL S CHAR(1320) VAR EXTERNAL;
                                        /*NOTE THAT S IS THE INPUT STRING
                                        HERE*/
        S=STRNG;
        DO I=1 TO L;
             CH=SUBSTR(S,I,1);
             IF 'A'<=CH & CH <='Z' THEN DO;
                  ALP(I)=T;
                  GO TO ENDLP;
                  END;
             IF ('0'<=CH & CH<='9') | (CH='.') THEN DO;
                  NUM(I)=T;
                  GO TO ENDLP;
                  END ;
             DO K=1 TO NOPS;
                  IF CH=OPN(K) THEN DO;
                       OPP(I)=T;
                       GO TO ENDLP;
                       END;
                  END;
             PUT LIST (' CHARACTER ',CH,' IN POSITION ',I,' OF STRING',
             S,' IS NOT A VALID FORTRAN CHARACTER');
             SIGNAL ERROR;
             STOP;
```

```
ENDLP   :     END;
        I=1;
STBIT:  IF OPP(I) THEN DO;
              DO J=I TO L WHILE (OPP(J));
                  OP(J)=T;
                  END;
              IF J>L THEN RETURN;
              I=J;
              END;
                                        /* ITH CHARACTER SHOULD BE FIRST
                                        NON-OPERATOR  */
        IF ALP(I) THEN VAR(I)=T;
        IF NUM(I) THEN CONST(I)=T;
        DO J=I TO L WHILE (OPP(J)=F);
              VAR(J)=VAR(I);
              CONST(J)=CONST(I);
              END;
        IF J>L THEN RETURN;
        IF CONST(I) & ALP(J-1) THEN DO;
              CH=SUBSTR(S,J-1,1);
              IF (CH¬='D') & (CH¬='E') THEN GO TO SCRUNCH;
              CH=SUBSTR(S,J,1);
              IF CH='+'|CH='-' THEN DO;
                  I=J+1;
                  CONST(J)=T;
                  IF I>L THEN RETURN;
                  GO TO STBIT;
                  END;
              END;
        I=J;
        GO TO STBIT;
SCRUNCH:PUT LIST(' CHARACTER ',CH ,' AFTER POSITION ',I,' IN STRING ',
        S,' IS NOT A VALID PART OF A FORTRAN CONSTANT');
        SIGNAL ERROR;
        STOP;
        END ANALYS;
GTNAM  :PROC;
        DCL S CHAR(1320) VAR EXTERNAL;
                                        /*NOTE THAT S IS THE INPUT STRING
                                        HERE*/
        DCL ISTR CHAR(4);
        I=1;
        NELTS=1;
        NCONS=0;
CKTYPE :IF OP(I) THEN DO;
             CH=SUBSTR(S,I,1);
             IF CH='*' THEN DO;
                  CH=SUBSTR(S,I+1,1);
                  IF CH='*' THEN DO;
                       ELT(NELTS)='**';
                       I=I+2;
                       END;
                  ELSE DO;
                       ELT(NELTS)='*';
                       I=I+1;
                       END;
                  END;
             ELSE DO;
                  IF CH='(' THEN SIG(NELTS)=T;
                  ELSE;
                  ELT(NELTS)=CH;
```

```
                I=I+1;
                END;
            END;
        ELSE DO;
            TEMP=(0)' ';
            DO J=I TO L WHILE(OP(J)=F);
                TEMP=TEMP||SUBSTR(S,J,1);
                END;
            SIG(NELTS)=T;
            IF CONST(I) THEN DO;
                IF NCONS ¬=0 THEN DO;
                    DO K=1 TO NCONS;
                        IF CONS(K)=TEMP THEN DO;
                            PUT STRING (ISTR) EDIT(K) (F(4));
                            ELT(NELTS)='C$'||ISTR;
                            GO TO SETI;
                            END;
                        END;
                    END;
                NCONS=NCONS+1;
                CONS(NCONS)=TEMP;
                PUT STRING (ISTR) EDIT(NCONS) (F(4));
                ELT(NELTS)='C$'||ISTR;
                END;
            IF VAR(I) THEN ELT(NELTS)=TEMP;
SETI    :   I=J;
            END;
        IF I>L THEN RETURN;
        NELTS=NELTS+1;
        GO TO CKTYPE;
        END GTNAM;
BACKUS: PROC(BNF);
        DCL S CHAR(1320) VAR EXTERNAL;
                                    /*NOTE THAT S IS THE INPUT STRING
                                    HERE*/
        DCL BNF(5*NELTS,2) CHAR(8) VAR, TAIL CHAR(1), NULL CHAR(1) VAR
        INIT((0)' '), OPR(4) CHAR(2) VAR INIT('+','*','**','#'),
        (I,J,K,L                                        ) FIXED BIN;
        I=1;
        J=1;  BNF(J,1)=NULL;  BNF(J,2)='=';
NXTELT: L=LENGTH(BNF(J,2));
        IF SIG(I) THEN
SIGFRM:     DO;
            TAIL=SUBSTR(BNF(J,2),L,1);
            IF TAIL='='|TAIL='(' THEN
PREQ:           DO; DO K=1 TO 3; J=J+1; BNF(J,1)=OPR(K);BNF(J,2)='(';
                END PREQ;
            ELSE DO;
                IF ELT(I)¬='(' THEN GO TO SCRUNCH;
                END;
            J=J+1; BNF(J,1)='#';  BNF(J,2)=ELT(I);
            I=I+1;
            IF I<=NELTS THEN GO TO NXTELT;
            DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
            LBNF=J;
            RETURN;
            END SIGFRM;
                                    /*LOOK FOR OPERATORS +,-,*,/,**
                                    EACH OF THESE OPERATORS MUST BE
                                    FOLLOWED BY A VARIABLE, CONSTANT,
                                    OR (     */
```

```
          IF ELT(I)='+'|ELT(I)='-' THEN
PLM:          DO;
              IF SIG(I+1)=F THEN GO TO SCRUNCH;
              TAIL=SUBSTR(BNF(J,2),L,1);
              IF TAIL¬='='&TAIL¬='(' THEN
ADPRN:             DO; DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')';
                   END ADPRN;
              J=J+1;  BNF(J,1)=ELT(I);  BNF(J,2)='(';
              J=J+1;  BNF(J,1)='*';     BNF(J,2)='(';
              J=J+1;  BNF(J,1)='**';    BNF(J,2)='(';
              J=J+1;  BNF(J,1)='#';     BNF(J,2)=ELT(I+1);
              I=I+2;
              IF I<=NELTS THEN GO TO NXTELT;
              DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              LBNF=J;
              RETURN;
              END PLM;
          IF ELT(I)='*'|ELT(I)='/' THEN
MD:           DO;
              IF SIG(I+1)=F THEN GO TO SCRUNCH;
              J=J+1;  BNF(J,1)=NULL;    BNF(J,2)=')';
              J=J+1;  BNF(J,1)=NULL;    BNF(J,2)=')';
              J=J+1;  BNF(J,1)=ELT(I);  BNF(J,2)='(';
              J=J+1;  BNF(J,1)='**';    BNF(J,2)='(';
              J=J+1;  BNF(J,1)='#';     BNF(J,2)=ELT(I+1);
              I=I+2;
              IF I<=NELTS THEN GO TO NXTELT;
              DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              LBNF=J;
              RETURN;
              END MD;
          IF ELT(I)='**' THEN
EXP:          DO;
              IF SIG(I+1)=F THEN GO TO SCRUNCH;
              J=J+1;  BNF(J,1)=NULL;    BNF(J,2)=')';
              J=J+1;  BNF(J,1)='**';    BNF(J,2)='(';
              J=J+1;  BNF(J,1)='#';     BNF(J,2)=ELT(I+1);
              I=I+2;
              IF I<=NELTS THEN GO TO NXTELT;
              DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              LBNF=J;
              RETURN;
              END EXP;
          IF ELT(I)=')' THEN
RPRN:         DO;
              DO K=1 TO 4; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              I=I+1;
              IF I<=NELTS THEN GO TO NXTELT;
              DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              LBNF=J;
              RETURN;
              END RPRN;
          IF ELT(I)=',' THEN
CMA:          DO;
              DO K=1 TO 4; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              J=J+1;  BNF(J,1)='#';     BNF(J,2)='(';
              I=I+1;
              IF I<=NELTS THEN GO TO NXTELT;
              DO K=1 TO 3; J=J+1; BNF(J,1)=NULL; BNF(J,2)=')'; END;
              LBNF=J;
              RETURN;
```

```
            END CMA;
                                        /* IF ALL TESTS ON ELT(I) ARE
                                        FALSE:  */
        PUT LIST( ELT(I),' IS AN UNDEFINED SYMBOL IN STRING ',S);
        SIGNAL ERROR;
        STOP;
SCRUNCH:PUT LIST(' INVALID FORTRAN AT ELEMENTS ',ELT(I),' AND ',
        ELT(I+1),' OF STRING ',S);
        SIGNAL ERROR;
        STOP;
        END BACKUS;
MKTRPL: PROC;
        DCL (LPSI,N,A,C,LK,K(LBNF)                          ) FIXED BIN;
        HEAD=NULL;  TAIL=NULL;
        N=1;  A,C,LK=0;
LP:     DO I=2 TO LBNF;
            IF BNF(I,2)='(' THEN
LPRN:           DO;
                                        /* SET LENGTH FOR PSI, DEPENDING
                                        ON VALUE OF N */
                IF N<=9   THEN DO; LPSI=1; GO TO ALL; END;
                IF N<=99  THEN DO; LPSI=2; GO TO ALL; END;
                IF N<=999 THEN DO; LPSI=3; GO TO ALL; END;
                LPSI=4;
ALL:            ALLOCATE TRPL;      CALL STPTRS;
                SC=C;     DI=BNF(I,1);   SV=F;
                PUT STRING(PSI) EDIT(N) (F(LPSI));
                                        /* SET COUNTERS FOR NEXT TRIPLE*/
                A=A+1; LK=LK+1; K(LK)=C; C=N; N=N+1;
                GO TO ENDLP;
                END LPRN;
                                        /* IF BNF(I,2)=')', NO TRIPLE IS
                                        ALLOCATED, BUT COUNTERS ARE SET*/
            IF BNF(I,2)=')' THEN
RPRN:           DO; C=K(LK); LK=LK-1; A=A-1; GO TO ENDLP; END RPRN;
                                        /* BNF(I,2) IS A NAME */
            LPSI=LENGTH(BNF(I,2));
            ALLOCATE TRPL;      CALL STPTRS;
            SC=C;  DI=BNF(I,1);  PSI=BNF(I,2);    SV=F;
ENDLP:      END LP;
                                        /* THE TRIPLES AS USED HERE ARE
                                        DEFINED IN PETER SHERIDAN'S PAPER
                                        THE ARITHMETIC TRANSLATOR-
                                        COMPILER OF THE IBM FORTRAN
                                        AUTOMATIC CODING SYSTEM,
                                        COMMUNICATIONS OF THE ACM, 1957*/
STPTRS: PROC;
        IF HEAD=NULL THEN
            DO;  HEAD=P;  P->UP=NULL;  END;
        ELSE
            DO;  P->UP=TAIL;  TAIL->DOWN=P;  END;
        P->DOWN=NULL;
        TAIL=P;
        RETURN;
        END STPTRS;
        RETURN;
        END MKTRPL ;
SHRINK: PROC;
                                        /*THIS ROUTINE SEARCHES THE LIST
                                        OF TRIPLES AND DROPS THOSE WHOSE
                                        SC VALVES APPEAR ONLY ONCE IN THE
```

```
                                        LIST IF THEIR OPERATION IS NOT -
                                        */
        DCL (PT1,PT2,PT3,PT4) PTR;
        PT1=TAIL;
LPPT2:  PT2=PT1;
INCPT2: PT2=PT2->UP;
        IF PT2¬=NULL THEN
CKSC:        DO;
             IF PT2->SC¬=PT1->SC THEN GO TO INCPT2;
                                        /* IF THE TWO SC VALVES ARE EQUAL,
                                        THE TRIPLES POINTED TO BY PT1 AND
                                        PT2 SHOULD BE SAVED */
             PT2->SV=T;   PT1->SV=T;
             GO TO INCPT2;
             END CKSC;
        IF PT1->SV=T THEN
NXTPT1:      DO;
INCPT1:      PT1=PT1->UP;
             IF PT1=NULL THEN RETURN;
             IF PT1->SV=T THEN GO TO INCPT1;
                                        /* IF SV WAS SET TRUE IN AN EARLIER
                                        SCAN, OMIT THE SCAN NOW */
             GO TO LPPT2;
             END NXTPT1;
        IF PT1->DI='- ' THEN GO TO INCPT1;
                                        /* DROP THE TRIPLE POINTED TO BY
                                        PT1 */
        PT3=PT1;  PT2=PT1->UP;
        IF PT2=NULL THEN RETURN;        /* DO NOT DROP THE FIRST TRIPLE*/
        PT2->PSI=PT1->PSI;
        PT4=PT1->DOWN;
        PT2->DOWN=PT4;
        IF PT4=NULL THEN TAIL=PT2;
        ELSE PT4->UP=PT2;
        PT1=PT2;  FREE PT3->TRPL;
        GO TO LPPT2;
        END SHRINK;
SORT:   PROC;
        DCL(PT1,PT2,PT3,PT4 ,PT5                                  )PTR;
        PT1=HEAD;
STPT1:  PT2=PT1->DOWN;
        IF PT2=NULL THEN RETURN;
        IF PT2->SC=PT1->SC THEN DO;  PT1=PT2;  GO TO STPT1;  END;
                                        /*SET PT1 TO LAST TRIPLE IN
                                        SEGMENT */
STPT2:  PT2=PT2->DOWN;
        IF PT2=NULL THEN DO;  PT1=PT1->DOWN;  GO TO STPT1;  END;
        IF PT1->SC¬=PT2->SC THEN GO TO STPT2;
                                        /*FOR EQUAL SCS, REARRANGE THE
                                        LIST SO THAT THE TRIPLE POINTED
                                        TO BY PT2 BECOMES THE NEXT ONE
                                        AFTER THE ONE POINTED TO BY PT1*/
        PT3=PT2->UP;  PT4=PT2->DOWN;  PT5=PT1->DOWN;
        IF PT4=NULL THEN TAIL=PT3;  ELSE PT4->UP=PT3;
        PT2->DOWN=PT5;  PT5->UP=PT2;
        PT3->DOWN=PT4;  PT1->DOWN=PT2;  PT2->UP=PT1;  PT1=PT2;
        GO TO STPT1;
        END SORT;
STORE:  PROC;
        DCL (PT1,PT2,PT3                                          ) PTR,
(IS,I,LN                                                        ) FIXED BIN,
        TMP CHAR(6),
```

```
        CH CHAR(1);
        PT1=HEAD;  NSEGS=1;
NEWSG:  IS=PT1->SC;
        IF IS<10   THEN DO;  LN=1;  GO TO FNDS;  END;
        IF IS<100  THEN DO;  LN=2;  GO TO FNDS;  END;
        IF IS<1000 THEN DO;  LN=3;  GO TO FNDS;  END;
        LN=4;
FNDS:   PUT STRING (TMP) EDIT(IS)(F(LN)); S(NSEGS)=SUBSTR(TMP,1,LN);
        LS(NSEGS)=1;
NXTOPR: IF PT1->DI='**' THEN OPR(NSEGS,LS(NSEGS))='**';
        ELSE OPR(NSEGS,LS(NSEGS))=SUBSTR(PT1->DI,1,1);
             DO I=1 TO 8;  CH=SUBSTR(PT1->PSI,I,1);
             IF CH¬=' ' THEN
             OPND(NSEGS,LS(NSEGS))=OPND(NSEGS,LS(NSEGS))||CH;
        END;
        PT2=PT1;  PT1=PT1->DOWN;  FREE PT2->TRPL;
        IF PT1=NULL THEN RETURN;
        IF PT1->SC=IS THEN DO; LS(NSEGS)=LS(NSEGS)+1; GO TO NXTOPR;END;
        ELSE DO; NSEGS=NSEGS+1;  GO TO NEWSG;  END;
        END STORE;
REP:    PROC(ST) CHAR(1320) VAR RECURSIVE;
        DCL ST CHAR(6) VAR,CH CHAR(1),STR CHAR(1320) VAR, (I,J,LSEG)
        FIXED BIN;
        CH=SUBSTR(ST,1,1);
                                        /*TEST WHEIHER ST IS A SEGMENT*/
        IF CH<'0' | CH> '9' THEN RETURN(ST);
                                        /* IF CH IS A NUMBER, ANALYZE TH E
                                        SEGMENT ST*/
        DO I=1 TO NSEGS WHILE(S(I)¬=ST); END;
        IF I>NSEGS THEN DO;
             PUT LIST (ST,' IS NOT ONE OF THE GIVEN SEGMENTS');
             SIGNAL ERROR;
             END;
        LSEG=LS(I);
        IF OPR(I,1)='#' THEN DO;
             STR=REP(OPND(I,1))||'('||REP( OPND(I,2));
             IF LSEG >2 THEN DO;
                  DO J=3 TO LSEG ;
                       STR=STR||','||REP(OPND(I,J));
                       END;
                  END;
             STR=STR||')';
             RETURN(STR);
             END;
        IF LSEG =1 THEN DO;
             IF OPR(I,1)='-' THEN RETURN ('(-('||REP(OPND(I,1))||'))');
             ELSE RETURN(REP(OPND(I,1)));
             END;
        IF OPR(I,1)='-' THEN STR='(-'||REP(OPND(I,1));
        ELSE STR='('||REP(OPND(I,1));
        DO J=2 TO LSEG ;
             STR=STR||OPR(I,J)||REP(OPND(I,J));
             END;
        STR=STR||')';
        RETURN(STR);
        END REP;
DRV:    PROC(ST,X) RECURSIVE CHAR(1320) VAR;
        DCL(ST, X) CHAR(6) VAR, (D,RS,RO) CHAR(1320) VAR, CH CHAR(1),
        (I,L) FIXED BIN;
        CH=SUBSTR(ST,1,1);
        IF '0'>CH|CH>'9'  THEN DO;
             IF ST=X THEN RETURN('1.');
```

```
            ELSE RETURN('0.');
            END;
        DO I=1 TO NSEGS WHILE(S(I)¬=ST);  END;
        IF I>NSEGS THEN DO;
            PUT LIST( ST, ' IS NOT ONE OF THE GIVEN SEGMENTS');
            SIGNAL ERROR; STOP; END;
        IF RFX(I)='0'B THEN RETURN('0.');
        L=LS(I);
        IF L=1 THEN DO;
            IF OPR(I,L)='-' THEN RETURN('-('||DRV(OPND(I,L),X)||')');
            ELSE RETURN(DRV(OPND(I,L),X));
            END;
        LS(I)=L-1;
        IF OPR(I,L)='+'|OPR(I,L)='-' THEN
        RETURN ('('||DRV(S(I),X)||')'||OPR(I,L)
        ||'('||DRV(OPND(I,L),X)||')');
        IF OPR(I,L)='*' THEN DO;
            RS=REP(S(I));
           D='(('||DRV(S(I),X)||')*'||REP(OPND(I,L))||'+'||RS
            ||'*('||DRV(OPND(I,L),X)||'))';
            RETURN(D);
            END;
        IF OPR(I,L)='/' THEN DO;
            RO=REP(OPND(I,L));
            RS=REP(S(I));
            D='(('||DRV(S(I),X)||')*'||RO||'-'||RS        ||'*('
            ||DRV(OPND(I,L),X)||'))/('||RO||'*'||RO||')';
            RETURN(D);
            END;
        IF OPR(I,L)='**' THEN DO;
                                        /*NOTE THE REVERSE ORDER, SINCE
                                        ** IS DONE FROM RIGHT TO LEFT*/
            RO=REP(OPND(I,1));  D=DRV(OPND(I,1),X);
                 DO K=2 TO L;
                 OPND(I,K-1)=OPND(I,K); OPR(I,K-1)=OPR(I,K);  END;
            RS=REP(S(I));
            D=RO||'**'||RS||'*ALOG('||RO||')*('||DRV(S(I),X)||')+'
            ||RS||'*'||RO||'**('||RS||'-1.)*('||D||')';
            RETURN(D);
            END;
        IF OPR(I,1)='#' THEN
FUNC:       DO;
            IF LS(I)=1 THEN
            RETURN(FNDFCN(OPND(I,1),OPND(I,2))||'*('||DRV(OPND(I,2),X)
            ||')');
            END FUNC;
        END DRV;
FNDFCN: PROC(LF,Z) CHAR(1320) VAR;
        DCL (LF,Z) CHAR(6)VAR;
                                        /*THIS IS A LIST OF DERIVATIVES
                                        OF FORTRAN LIBRARY FUNCTIONS    */
        IF LF='EXP' THEN RETURN('EXP('||REP(Z)||')');
        IF LF='SIN' THEN RETURN('COS('||REP(Z)||')');
        IF LF='COS' THEN RETURN('(-SIN('||REP(Z)||'))');
        IF LF='SQRT' THEN RETURN('(.5/SQRT('||REP(Z)||'))');
        IF LF='TAN' THEN RETURN('(1.0/COS('||REP(Z)||')**2)');
        IF LF='COTAN' THEN RETURN('(-1.0/SIN('||REP(Z)||')**2)');
        IF LF='ALOG' THEN RETURN('(1.0/'||REP(Z)||')');
        IF LF='ALOG10' THEN RETURN('(0.4342945/'||REP(Z)||')');
        IF LF='ARSIN' THEN RETURN('(1.0/SQRT(1.0-'||REP(Z)||'**2))');
        IF LF='ARCOS' THEN RETURN('(-1.0/SQRT(1.0-'||REP(Z)||'**2))');
        IF LF='ATAN' THEN RETURN('(1.0/(1.0+'||REP(Z)||'**2))');
```

```
        IF LF='TANH' THEN RETURN('(1.0/COSH('||REP(Z)||')**2)');
        IF LF='SINH' THEN RETURN('COSH('||REP(Z)||')');
        IF LF='COSH' THEN RETURN('SINH('||REP(Z)||')');
        IF LF='ERF' THEN RETURN('(1.1283792*EXP(-('||REP(Z)||'**2)))');
        IF LF='ERFC' THEN
        RETURN('(-1.1283792*EXP(-('||REP(Z)||'**2)))');
        PUT LIST (LF,' IS AN UNDEFINED FUNCTION');  SIGNAL ERROR;
        END FNDFCN;
RPLCNS: PROC(STR);
        DCL (STR,STRA,STRB) CHAR(1320) VAR,DGT CHAR(8) VAR,CH CHAR(1),
        (INDC,INDST,I,J,LN,L                                  ) FIXED BIN;
        INDC=INDEX(STR,'C$');  IF INDC=0 THEN RETURN;
FNDDGT: INDST=INDC+2;  DGT=(0)' ';  LN=LENGTH(STR);
              DO J=INDST TO LN;  CH=SUBSTR(STR,J,1);
              IF '0'<=CH & CH<='9' THEN DO; DGT=DGT||CH;GO TO ELP;END;
              IF CH¬=' ' THEN GO TO FNDI;
ELP:          END;
FNDI:   DGT=' '||DGT||' ';
        GET STRING (DGT) LIST(I);
        STRA=SUBSTR(STR,1,INDC-1)||CONS(I);
        L=LN-J+1;
        IF L<=0 THEN DO; STR=STRA;  RETURN;  END;
        STRB=SUBSTR(STR,J,L);  STR=STRA||STRB;
        INDC=INDEX(STRB,'C$');  IF INDC=0 THEN RETURN;
        INDC=INDC+LENGTH(STRA);
        GO TO FNDDGT;
        END RPLCNS;
SIMP:   PROC;
        DCL(PT1,PT2                                           )PTR,
        (K                                                   )FIXED BIN,
        (SMDRP,
        LN1, IFDRP)BIT(8) INIT('0'B),(TST,ONE INIT(1.0)) FLOAT BIN,
        MKCONS RETURNS(CHAR(8));
        PT1=TAIL;
HDCK:   IF PT1=HEAD THEN GO TO LST;
                                  /*DO NOT DROP AN INITIAL SEGMENT
                                  OF LENGTH ONE                  */
FCONS:  IF SUBSTR(PT1->PSI,1,2)¬='C$' THEN GO TO LNS;
                                  /*FIND SUBSCRIPT IN CONS       */
                                  /*CONVERT TO FLOAT BIN         */
        GET STRING(PT1->PSI) EDIT(K)(X(2),F(4));  TST=CONS(K);
        IF TST=0.0 THEN
ZRO:          DO;
              IF PT1->DI='+'|PT1->DI='-' THEN
PLM:               DO;
DRP:               CALL CKLN; IF LN1=F THEN
                        DO;CALL DTRPL; IF PT1=NULL THEN GO TO LST;
                                  /*IF THE SEGMENT IS NOT OF LENGTH
                                  1, THE INITIAL TRIPLE MAY BE
                                  DROPPED FROM THIS SEGMENT      */
                        GO TO FCONS; END;
                   IF PT1=HEAD THEN GO TO LST;
                                  /*HERE THE LENGTH IS 1  */
                   CALL RPLRF; CALL DTRPL; GO TO HDCK;
                   END PLM;
              IF PT1->DI='*' THEN
                   DO;CALL DSG;IF PT1=NULL THEN GO TO LST;
                   GO TO HDCK; END;
              IF PT1->DI='**' THEN
EXP:               DO; IF PT1=HEAD THEN DO; CALL DSG; GO TO LST;END;
                   PT2=PT1->UP; IF PT2->SC=PT1->SC THEN
```

```
                    DO; PT2->PSI=MKCONS(ONE); CALL DRST;
                    IF PT1=HEAD THEN GO TO LST;ELSE GO TO EX1;
                    END;
                CALL DSG; IF PT1=NULL THEN GO TO LST;GO TO HDCK;
                END EXP;
            END ZRO;
        IF TST=1.0 THEN
ONETST:     DO; IF PT1->DI='/' THEN GO TO DRP;
            IF PT1->DI='*' THEN
MCK:            DO; CALL CKLN;IF LN1 THEN GO TO L1;
                                /*TEST IF PT1 IS FIRST IN SEGMENT
                                                                  */
                IF PT1¬=HEAD THEN
FRSTCK:             DO;PT2=PT1->UP;
                    IF PT2->SC=PT1->SC THEN GO TO DRP;
                    END FRSTCK;
                                /*PT2 BELOW POINTS TO A TRIPLE IN
                                THE SAME SEGMENT AS PT1, SINCE
                                PT1 GIVES THE FIRST TRIPLE IN A
                                SEGMENT OF LENGTH MORE THAN 1  */
                PT2=PT1->DOWN;
                                /*DO NOT ELIMINATE A FIRST TRIPLE
                                INDICATING AN OPERATION 1/A    */
                IF PT2->DI='/' THEN GO TO OPCK; ELSE GO TO DRP;
                END MCK;
            IF PT1->DI='**' THEN
EX1:            DO;IF PT1=HEAD THEN DO; CALL DSG; GO TO LST; END;
                PT2=PT1->UP;
                IF PT2->SC=PT1->SC THEN CALL DRST; ELSE CALL DSG;
                GO TO HDCK;
                END EX1;
            END ONETST;
OPCK:   CALL CKOPN; IF IFDRP THEN DO; IFDRP=F; GO TO FCONS;   END;
LNS:    CALL CKLN; IF LN1 THEN
L1:         DO;IF PT1=HEAD THEN GO TO LST;
            IF PT1->DI='-' THEN GO TO NXT;
            CALL RPLRF; CALL DTRPL; GO TO HDCK;
            END L1;
NXT:    PT1=PT1->UP; IF PT1¬=NULL THEN GO TO FCONS;
LST:    IF SMDRP THEN DO;PT1=TAIL;SMDRP=F;GO TO HDCK;END;
        RETURN;
MKCONS: PROC(VAL) CHAR(8);
        DCL (VAL FLOAT,I FIXED) BIN,CSTR CHAR(8),ISTR CHAR(6);
        DCL VTST FLOAT BIN,STST CHAR(20);
             DO I=1 TO NCONS;  VTST=CONS(I);
             IF ABS(VTST-VAL)<=.5E-6*VAL THEN GO TO TSTI;
             END;
TSTI:   IF I<=NCONS THEN PUT STRING(ISTR) EDIT(I) (     F(4));
        ELSE
             DO;NCONS=NCONS+1;PUT STRING(ISTR) EDIT(NCONS)(     F(4));
             CONS(NCONS)=(20)' ';
             PUT STRING(CONS(NCONS)) EDIT(VAL)(E(20,7));
             END;
        CSTR='C$'||ISTR;
        RETURN (CSTR);
        END MKCONS;
DTRPL:  PROC;
        DCL (PTU,PTD                                          )PTR;
        SMDRP=T;
        PTU=PT1->UP; PTD=PT1->DOWN; FREE PT1->TRPL;
        IF PTU=NULL THEN DO; HEAD=PTD;PT1=NULL;PTD->UP=PTU;RETURN;END;
        IF PTD=NULL THEN TAIL=PTU; ELSE PTD->UP=PTU;
```

```
        PTU->DOWN=PTD;  PT1=PTU; RETURN;
        END DTRPL;
DRST:   PROC;
                                        /*THIS ROUTINE FREES TRIPLES FROM
                                        PT1 DOWN TO THE NEXT SEGMENT*/
        DCL(PT2,PDRP,PKP                                        )PTR,
        SCTST FIXED BIN;
        SMDRP=T;
        PKP=PT1->UP; SCTST=PT1->SC; PDRP=PT1;
FRTR:   PT2=PDRP->DOWN; FREE PDRP->TRPL;
        IF PT2=NULL THEN
             DO; TAIL=PKP; PKP->DOWN=PT2; PT1=PKP; RETURN; END;
        IF PT2->SC=SCTST THEN DO; PDRP=PT2; GO TO FRTR; END;
                                        /*RESET POINTERS                */
        PT2->UP=PKP;
        IF PKP=NULL THEN HEAD=PT2; ELSE PKP->DOWN=PT2;
        PT1=PKP;
        RETURN;
        END DRST;
DSG:    PROC;
        DCL (PT2,PKP) PTR,SCTST FIXED BIN;
        SCTST=PT1->SC;   PKP=PT1;
SPKP:   PKP=PKP->UP;
                                        /*IF THIS IS THE FIRST SEGMENT,
                                        KEEP THE FIRST TRIPLE  */
        IF PKP=NULL THEN
             DO; HEAD->PSI=PT1->PSI;
             CALL CKLN; IF LN1 THEN DO; PT1=NULL; RETURN; END;
             PT1=HEAD->DOWN; GO TO DRPSG; END;
        IF PKP->SC=SCTST THEN GO TO SPKP;
        PT2=PKP->DOWN; PT2->PSI=PT1->PSI; PT1=PT2; CALL RPLRF;
DRPSG:  CALL DRST;
        RETURN;
        END DSG;
RPLRF:  PROC;
        DCL PT2 PTR,(NSG,SCTST) FIXED BIN,CH CHAR(1);
                                        /*REPLACE REFERENCES TO A SEGMENT
                                        PT1->SC BY PT1->PSI           */
        PT2=PT1->UP;  SCTST=PT1->SC;
CKPSI:  CH=SUBSTR(PT2->PSI,1,1); IF '0'<=CH&CH<='9' THEN
             DO;  GET STRING(PT2->PSI) LIST(NSG);
             IF NSG=SCTST THEN PT2->PSI=PT1->PSI;
             END;
        PT2=PT2->UP; IF PT2¬=NULL THEN GO TO CKPSI;
        RETURN;
        END RPLRF;
CKLN:   PROC;
        LN1=T;
        IF PT1=TAIL THEN GO TO CKUP;
        PT2=PT1->DOWN;
        IF PT2->SC=PT1->SC THEN DO; LN1=F; RETURN; END;
CKUP:   IF PT1=HEAD THEN RETURN;
        PT2=PT1->UP; IF PT2->SC=PT1->SC THEN LN1=F;
        RETURN;   END CKLN;
CKOPN:  PROC;
        DCL (OP1,OP2) CHAR(2),(C2 FLOAT, K2 FIXED) BIN, PTST PTR;
        PT2=PT1->UP;
STPT2:  IF PT2=NULL THEN RETURN;
        IF PT2->SC¬=PT1->SC THEN RETURN;
        IF SUBSTR(PT2->PSI,1,2)¬='C$' THEN
             DO; IF PT2->DI='**' THEN  RETURN;
```

```
                                        /*NOTE ** IS NOT COMMUTATIVE    */
                ELSE DO; PT2=PT2->UP; GO TO STPT2; END;
            END;
        OP1=PT1->DI; OP2=PT2->DI;
        GET STRING(PT2->PSI) EDIT(K2) (X(2),F(4)); C2=CONS(K2);
                                        /*IF ADDITION OR SUBTRACTION OF
                                        CONSTANTS IS INDICATED, PERFORM
                                        THE OPERATION                   */
        IF OP1='-' THEN DO; TST=-TST; GO TO TSTSB;  END;
        IF OP1¬='+' THEN GO TO MDTST;
TSTSB:  IF OP2='-' THEN C2=-C2;
        C2=C2+TST;
        IF C2<0.0 THEN DO; PT2->DI='-'; C2=-C2;END;ELSE PT2->DI='+';
        GO TO DRPTR;
                                        /* IF * OR / IS INDICATED,
                                        PERFORM THE OPERATION           */
MDTST:  IF OP1='/' THEN DO; TST=1.0/TST; GO TO DOP2; END;
        IF OP1¬='*' THEN GO TO EXPTST;
DOP2:   IF OP2='/' THEN C2=1.0/C2;
        C2=C2*TST; PT2->DI='*'; GO TO DRPTR;
EXPTST: IF OP1¬='**' THEN  RETURN;
                                        /* IF THE LAST OPERATION IN A
                                        SEGMENT IS CONSTANT**CONSTANT,
                                        PERFORM IT                      */
        PTST=PT1->DOWN;
        IF PTST->SC=PT1->SC THEN RETURN;
        C2=C2**TST;
DRPTR:  CALL DTRPL; IFDRP=T;
                                        /*REPLACE PT2->PSI BY NEW VALUE*/
        PT2->PSI=MKCONS(C2);
        RETURN;
        END CKOPN;
        END SIMP;
COUNT:  PROC;
        DCL (LSS,SCTST) FIXED BIN, PT1 PTR;
        PT1=HEAD; NSEGS=1; LMAX=0;
NWSG:   LSS=1; SCTST=PT1->SC;
NWPT:   PT1=PT1->DOWN; IF PT1=NULL THEN RETURN;
        IF PT1->SC=SCTST THEN DO; LSS=LSS+1;GO TO NWPT; END;
        NSEGS=NSEGS+1; IF LSS>LMAX THEN LMAX=LSS ; GO TO NWSG;
        END COUNT;
CKRF:   PROC;
ALLOCATE RFX(NSEGS) BIT(1); RFX='0'B;
            DO I=NSEGS TO 1 BY -1;
                DO J=1 TO LS(I);        /*EXAMINE OPERANDS IN THE ITH
                                        SEGMENT                         */
                IF OPND(I,J)=X THEN DO;RFX(I)='1'B;GO TO NXTI;END;
                CH=SUBSTR(OPND(I,J),1,1);
                IF '0'<=CH & CH <='9' THEN
                                        /*SEE IF THE SEGMENT OPND(I,J)
                                        REFERS TO X                     */
                    DO;DO K=1 TO NSEGS WHILE(OPND(I,J)¬=S(K));END;
                    IF RFX(K) THEN DO;RFX(I)='1'B;GO TO NXTI;END;
                    END;
                END;
NXTI:       END;
        RETURN;
        END CKRF;
        END DERV;
```

# APPENDIX D

## Listings of Cataloged Procedures VMMCLG and VMMLG

### VMMCLG

```
//     PROC VMMREGN=350K,GOREGN=350K,EDTOPTS='LIST,MAP',EDTREGN=260K,
//              RUNREGN=260K,REGN=260K,OPTIONS=,LSIZE='(232K,100K)'
//VMMTR  EXEC  PGM=IEFBR14,REGION=&VMMREGN
//PRGG DD DSNAME=C145.B12533.T7120.SYSLMOD(VMMTR),DISP=SHR
//GO  EXEC  PGM=*.VMMTR.PRGG,REGION=&GOREGN
//SYSPRINT  DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
//SYSPNCH DD   UNIT=(CTC,,DEFER),DCB=(RECFM=F,BLKSIZE=80)
//RDFILE DD DSNAME=C145.B12533.T7120.SOURCE(SKFCN),DISP=SHR
//CDFILE DD DSNAME=&&PDAT,DISP=(NEW,PASS,DELETE),
//              SPACE=(CYL,(1,1),RLSE),
//  UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=7200)
//PROG DD DSNAME=&&FORT,DISP=(NEW,PASS,DELETE),SPACE=(CYL,(1,1),RLSE),
//              UNIT=(DISK,SEP=CDFILE),DCB=*.CDFILE
//FTH   EXEC PGM=FORTRANH,REGION=&REGN,PARM='&OPTIONS',COND=(5,LT,GO)
//SYSPUNCH DD UNIT=(CTC,,DEFER),DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//SYSPRINT  DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
//SYSLIN DD DSNAME=&&SYSLIN,DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200),
//  DISP=(NEW,PASS,DELETE),SPACE=(3200,(40,10),RLSE),UNIT=2314
//SYSUT1 DD DSN=&&SYSUT1,SPACE=(CYL,(1,1)),UNIT=(DISK,SEP=SYSPRINT)
//SYSIN DD DSNAME=&&FORT,DISP=(OLD,DELETE,DELETE)
//        DD DDNAME=CARDS
//EDT      EXEC PGM=LINKEDIT,COND=(5,LT,FTH),
//              PARM='&EDTOPTS,SIZE=&LSIZE',REGION=&EDTREGN
//SYSPRINT DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBA,LRECL=121,BLKSIZE=726)
//SYSLIN    DD DSNAME=*.FTH.SYSLIN,DISP=(OLD,DELETE)
//     DD DSNAME=C145.B12533.T7120.EDT,DISP=SHR
//           DD DDNAME=SYSIN
//SYSLIB DD DSNAME=SYS1.AMDLIB,DISP=SHR
//        DD DSNAME=SYS1.FORTLIB,DISP=SHR
//SYSUT1 DD SPACE=(3072,(20,10),RLSE),UNIT=DISK,DCB=BLKSIZE=6144
//SYSLMOD DD UNIT=DISK,SPACE=(3072,(50,10,1),RLSE),DISP=(,PASS),
//              DSNAME=&&G(DVDN)
//LIB DD DSNAME=C145.B12533.T7120.SYSLMOD,DISP=SHR
//RUN      EXEC PGM=*.EDT.SYSLMOD,COND=((5,LT,FTH),(5,LT,EDT)),
//              REGION=&RUNREGN
//CDFILE   DD DSNAME=&&DATA,DISP=(,,DELETE),UNIT=2314,
//     SPACE=(CYL,(1,1),RLSE),DCB=(RECFM=FB,LRECL=80,BLKSIZE=7200)
//FT06F001 DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
//FT07F001 DD UNIT=(CTC,,DEFER),DCB=(RECFM=F,BLKSIZE=80)
//SYSPRINT DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBA,LRECL=133,BLKSIZE=798)
//FT05F001 DD DSNAME=*.CDFILE,VOLUME=REF=*.CDFILE,
//  DISP=(OLD,DELETE,DELETE)
//TEMP DD DSNAME=&&TDAT,DISP=(,,DELETE),UNIT=(2314,SEP=CDFILE),
//     SPACE=(CYL,(1,1)),DCB=*.CDFILE
//XTFILE DD DSNAME=&&XDAT,DISP=(,,DELETE),UNIT=(2314,SEP=CDFILE),
//       SPACE=(CYL,(1,1),RLSE),DCB=(RECFM=F,BLKSIZE=80)
//FT01F001 DD DSNAME=*.XTFILE,VOLUME=REF=*.XTFILE,
//  DISP=(OLD,DELETE,DELETE)
//STOR DD DSNAME=&&SDAT,DISP=(,,DELETE),UNIT=2314,
//              SPACE=(CYL,(1,1)),DCB=*.CDFILE
//SYSIN DD DSNAME=&&PDAT,DISP=(OLD,DELETE,DELETE),
//    VOLUME=REF=*.GO.CDFILE
//*     IF TROUBLE,SEE M. GABRIEL, 221-B227
```

VMMLG

```
//     PROC  EDTOPTS='LIST,MAP',EDTREGN=260K,LSIZE='(232K,100K)',        C00000010
//              GOREGN=260K                                               00000020
//EDT  EXEC  PGM=LINKEDIT,PARM='&EDTOPTS,SIZE=&LSIZE',REGION=&EDTREGN    00000030
//LIB DD DSNAME=C145.B12533.T7120.SYSLMOD,DISP=SHR                        00000040
//SYSPRINT DD UNIT=(CTC,,DEFER),DCB=(LRECL=121,BLKSIZE=1573)              00000050
//SYSUT1 DD SPACE=(3072,(20,10),RLSE),UNIT=DISK,DCB=BLKSIZE=6144          00000060
//SYSLIB DD DSNAME=SYS1.AMDLIB,DISP=SHR                                   00000070
//        DD DSNAME=SYS1.FORTLIB,DISP=SHR                                 00000080
//SYSLMOD DD UNIT=DISK,SPACE=(3072,(50,50,1),RLSE),DISP=(MOD,PASS),       C00000090
//              DSNAME=&&G(G),DCB=BLKSIZE=6144                            00000100
//SYSLIN DD DSNAME=C145.B12533.T7120.EDT,DISP=SHR                         00000110
//        DD DDNAME=SYSIN                                                 00000120
//GO EXEC PGM=*.EDT.SYSLMOD,COND=(5,LT,EDT),REGION=&GOREGN                00000130
//FT06F001 DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBM,LRECL=133,BLKSIZE=1596)    00000140
//FT07F001 DD UNIT=(CTC,,DEFER)                                           00000150
//CDFILE DD DSN=&&DATA,UNIT=DISK,SPACE=(CYL,(1,1),RLSE),                  C00000160
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=7200),DISP=(,,DELETE)      00000170
//TEMP   DD DSN=&&TDAT,UNIT=(2314,SEP=CDFILE),SPACE=(CYL,(1,1)),          C00000180
//              DISP=(,,DELETE),DCB=*.CDFILE                              00000190
//XTFILE DD DSN=&&XDAT,UNIT=(2314,SEP=CDFILE),SPACE=(CYL,(1,1),RLSE),     C00000200
//              DISP=(,,DELETE),DCB=(RECFM=F,BLKSIZE=80)                  00000210
//STOR DD DSNAME=&&SDAT,DISP=(,,DELETE),UNIT=2314,                        C00000220
//              SPACE=(CYL,(1,1)),DCB=*.CDFILE                            00000230
//FT05F001 DD DSN=*.CDFILE,VOL=REF=*.CDFILE,DISP=(OLD,DELETE,DELETE)      00000240
//FT01F001 DD DSN=*.XTFILE,VOL=REF=*.XTFILE,DISP=(OLD,DELETE,DELETE)      00000250
//SYSPRINT DD UNIT=(CTC,,DEFER),DCB=(RECFM=FBM,LRECL=121,BLKSIZE=1573)    00000260
//*    IF TROUBLE,SEE M. GABRIEL, 221-B227                                00000270
```

# APPENDIX E

## Sample Function and Its Derivatives

```
 FUNCTION IS
  A1*EXP(((-B1 +X)/C1 )**2)+
 A2 *EXP(((-B2 +X)/C2 )**2)+
 A3 *EXP(((-B3 +X)/C3 )**2)+
 A4 *EXP(((-B4 +X)/C4 )**2)+
 A5 *EXP(((-B5 +X)/C5 )**2)+
 A6 *EXP(((-B6 +X)/C6 )**2)+
 A7 *EXP(((-B7 +X)/C7 )**2)+
 A8 *EXP(((-B8 +X)/C8 )**2)+
 A9 *EXP(((-B9 +X)/C9 )**2)+
 A10*EXP(((-B10+X)/C10)**2)


 DERIVATIVE OF FUNCTION WITH RESPECT TO A1 IS
EXP((((-B1+X)/C1)**2))


 DERIVATIVE OF FUNCTION WITH RESPECT TO B1 IS
(A1*(EXP((((-B1+X)/C1)**2))*(2*((-B1+X)/C1)*(((-(1.))*C1)/(C1*C1)))))


 DERIVATIVE OF FUNCTION WITH RESPECT TO C1 IS
(A1*(EXP((((-B1+X)/C1)**2))*(2*((-B1+X)/C1)*((-((-B1+X)))/(C1*C1)))))
```

## ACKNOWLEDGMENT

I wish to thank J. R. Gabriel for suggesting the use of Sheridan's methods for syntax analysis.

## REFERENCES

1. Gabriel, Marian, *Special-purpose Language for Least Squares Fits*, ANL-7495 (Sept 1968).

2. Davidon, William C., *Variable Metric Method for Minimization*, ANL-5990 (Rev. 2) (Feb 1966).

3. Sheridan, Peter B., *The Arithmetic Translator-Compiler of the IBM FORTRAN Automatic Coding System*, Comm. ACM. *2* (Feb 1959), pp. 9-21.

4. Garbow, Burton S., *Variable Metric Minimization*, Argonne National Laboratory, Applied Mathematics Division Internal Report Program Z013 (unpublished), August 9, 1965.

ARGONNE NATIONAL LAB WEST
3 4444 00008196 8