Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

THE SAC-1 POLYNOMIAL SYSTEM*

by

George E. Collins

Technical Report #115†

Computer Sciences Department and Computing Center
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin

Technical Reference 2: 1968

January 1968

Revised March 1968

*The research described in this report was supported by the University of Wisconsin Computing Center.

†This report is also being distributed as University of Wisconsin Computing Center Technical Report #2 (first edition January 1968, second edition March 1968, third edition March 1971).

TABLE OF CONTENTS

1. Introduction

SAC-1 is a programming system for symbolic and algebraic calculations. SAC-1 is organized into a hierarchy of subsystems. At the base of this hierarchy is the SAC-1 subsystem for list processing, which was described in a previous paper [1]. The SAC-1 subsystem for infinite-precision integer arithmetic, which depends on and utilizes the list processing subsystem, has also been previously documented [2]. The present paper describes the SAC-1 subsystem for formal operations on multivariate polynomials with infinite-precision integer coefficients, and assumes familiarity with the two previous documents.

The PM system [3] was likewise organized as a hierarchy of subsystems and, in fact, the three SAC-1 subsystems just mentioned correspond to the three PM subsystems. The main difference is that while PM was programmed for a particular computer (the IBM 7094), SAC-1 can be easily made operational on any general purpose digital computer for which an A.S.A. FORTRAN compiler is available. In order to implement SAC-1 on such a computer it is only necessary to (1) compile the SAC-1 FORTRAN subprograms (listings of which are supplied in the appendices of the SAC-1 papers) and (2) program in machine language the few simple SAC-1 "primitive" subprograms (according to the specifications in [1] and [2]). The effort required to implement SAC-1 should in most cases amount to no more than one man-month. At the time of this

writing, the two previous SAC-1 subsystems have been, or are being, implemented on at least seven different computers at six institutions.

Several additional SAC-1 subsystems are planned. Work is currently in progress on a subsystem for operations on rational functions (regarded as relatively prime pairs of multivariate polynomials, and including infinite-precision rational numbers as a special case). Other subsystems will provide operations on truncated power series with polynomial or rational function coefficients and methods for the exact solution of systems of linear equations with polynomial or rational function coefficients. Longer range plans include capabilities for symbolic processing of algebraic and transcendental functions.

Besides being very easy to implement on almost any computer, one of the design objectives of SAC-1 is a system that is highly efficient with respect to execution time. This aspect of PM was discussed in [4], where comparisons were made between PM, ALPAK [5] and FORMAC [6]. PM's efficiency resulted from several factors, including the use of the recursive canonical form for polynomials, the use of a reference count list processing system, and the use of assembly language programming in critical parts of the system. All these factors are fully retained in SAC-1 except for the last. In Section 6 we assess the extent to which this has impaired the efficiency of SAC-1 relative to PM . It is shown that this efficiency factor is probably no more than 3.0 and can easily be reduced to about 1.5 with only a little additional effort in implementing SAC-1.

For the most part, the SAC-1 polynomial subsystem is just a translation of the PM polynomial subsystem into A.S.A. FORTRAN [10]. However, a few improvements have been made. The list representation of a polynomial has been changed to reduce somewhat the memory requirements. A subroutine, PERASE, which erases the list representing an arbitrary polynomial, has been included, thereby exploiting one of the ways in which a reference count system can improve efficiency. The polynomial product algorithm has been modified so as to reduce the required number of list inversions. Polynomial variables may now consist of an arbitrary number of characters, and they are now uniquely represented as lists in computer memory much the same as are atomic symbols in LISP [7]. The biggest change pertains to polynomial substitution and reordering the variables of a polynomial. Whereas in PM reordering of variables was achieved as a byproduct of substitution, in SAC-1 reordering is achieved through an independent mechanism. As a result, the required substitution mechanism is logically simplified, is easier to apply, and is likely more efficient. All these changes are discussed in more detail in Sections 2, 3 and 4 .

2. Internal and External Representations of Polynomials

All SAC-1 polynomials are represented in recursive canonical form, both internally and externally. This terminology is carried over from the PM system [3], and refers to the fact that a polynomial in n variables is always regarded as a polynomial in one variable (called the main variable) whose coefficients are themselves polynomials in $n - 1$ variables. As a special case, any (infinite-precision) integer is a polynomial in zero variables. This implies an assumed ordering of the variables of any polynomial. Whenever we write $P(x_1, \ldots, x_n)$, displaying the variables of P, the intention is to specify this ordering, x_n being the main variable, x_{n-1} being the main variable of the coefficients of P, etc.

If $P(x_1, \ldots, x_n)$ is any non-zero polynomial in $n \geq 1$ variables, then we have uniquely $P(x_1, \ldots, x_n) = \sum_{i=1}^{r} P_i(x_1, \ldots, x_{n-1}) \cdot x_n^{e_i}$, where $e_1 > e_2 > \ldots > e_r \geq 0$ and the $P_i(x_1, \ldots, x_{n-1})$ are all non-zero.

A polynomial variable is any sequence of letters and decimal digits whose first character is a letter (there is no restriction on the number of characters).

The external canonical form of a polynomial is a sequence of characters. Suppose $n = 1$ and x_1 is ALPHA.

```
(+1ALPHA**3-37ALPHA**1+239ALPHA**0)
```

is an example of a polynomial in external canonical form. If Π_i is the

representation of P_i as a signed decimal integer, ϵ_i is the representation of e_i as an unsigned decimal integer, and x_1 is the sequence of characters ξ, the general form is

$$(\Pi_1\xi^{**}e_1\Pi_2\xi^{**}e_2 \ldots \Pi_r\xi^{**}e_r)$$

Polynomials are invariably output in this form. The only deviation permitted from this canonical form in input polynomials is that the sign of the leading coefficient may be omitted if positive.

Now suppose $n > 1$.

```
((+1ALPHA**3-37ALPHA**1+239ALPHA**0)BETA**4+(-29ALPHA**2-512ALPHA**1)
  BETA**2))
```

is an example of a polynomial in two variables written in external canonical form (BETA is the main variable). In general, for $n > 1$, if Π_i is the external canonical form for $P_i(x_1, \ldots, x_{n-1})$, x_n is the sequence of characters ξ, and ϵ_i is defined as above, then the external canonical form of $P(x_1, \ldots, x_n)$ is

$$(\Pi_1\xi^{**}e_1 + \Pi_2\xi^{**}e_2 \ldots\ldots + \Pi_r\xi^{**}e_r)$$

The zero polynomial is always a polynomial in zero variables and hence its external canonical form is `+0` (the `+` may be omitted in input).

Notice that variables may occur vacuously in a polynomial. For example, the following is permissible (and useful):

```
((+1ALPHA**0)BETA**0)
```

For further examples of polynomials in external canonical form, see Section 5.

Internally, the recursive canonical form of a polynomial is represented as a list of order $n + 1$, where n is the number of variables.

The zero polynomial is represented as the null list. A polynomial in zero variables is an infinite-precision integer and as such is represented as a first order list (see [2]). Assume, inductively, that the polynomials $P_i(x_1, \ldots, x_{n-1})$ are represented by lists Π_i. The variable x_n is represented as a first order list ξ (explained below). Then $P(x_1, \ldots, x_n)$ is represented as the list

$$(\xi, \Pi_1, e_1, \Pi_2, e_2, \ldots, \Pi_r, e_r)$$

In this list the exponents e_i are atoms -- FORTRAN integers -- not infinite precision integers.

In PM the list representing $P(x_1, \ldots, x_n)$ was of the form $((\Pi_1, e_1^*), (\Pi_2, e_2^*), \ldots, (\Pi_r, e_r^*))$, where e_i^* was an atom representing both e_r and the variable x_n. The SAC-1 list representation has the advantage of requiring somewhat less memory. For example, if $P(x_1, \ldots, x_n)$ is of degree r in each of its variables, has no "missing terms", and all of its integer coefficients are K-precision, then the SAC-1 representation requires approximately $(K + 2)r^n$ cells as compared with approximately $(K + 3)r^n$ for PM. This change probably also reduces execution time somewhat, but this is not entirely evident since it has effects in both directions.

Variables are represented internally as infinite-precision integers. If a variable v is the sequence of characters $\alpha_1 \alpha_2 \ldots \alpha_n$ and if N_i is the SAC-1 internal code for α_i, then v is represented by the infinite-precision integer $N = \sum_{i=1}^{n} N_i \cdot 64^{n-i}$.

Variables are stored uniquely in the computer memory as are atomic symbols in LISP [7]. There is a symbol list whose elements are the infinite-precision integers representing all the variables occurring in polynomials which have been read in. The location of the symbol list is kept in the symbolic location SYMLST, which is the first item in the labelled common block TR2. When a variable has been read which should occur as the first element of the list for a polynomial, the variable is presented to a subprogram PROSYM. PROSYM converts the variable to an integer and searches the symbol list for an occurrence of this integer. If found, the location of the occurrence on the symbol list is returned. Otherwise, the integer is prefixed to the symbol list. This is explained in more detail in Section 3.

3. Descriptions of the Subprograms

There are 36 subprograms in the SAC-1 polynomial system. In this section we describe the function and effect of each subprogram. It is designed to enable one to use the system. Section 4 describes the algorithms employed by the subprograms. It explains how the subprograms achieve their effects.

The subprograms are grouped below according to their use and purpose into the following categories: input-output, arithmetic, differentiation, greatest common divisors, substitution and reordering of variables, and miscellaneous. Not all the subprograms are likely to be used in applications programs. Some exist chiefly because they were useful in implementing the major subprograms, or because they will be useful in implementing later SAC-1 subsystems. Or, they may be useful to someone who wishes to construct his own extension of the system for some specialized problem. These subprograms are marked below with an asterisk and may be ignored in a first reading.

In order to facilitate the following descriptions, we first establish some conventions and terminology.

A proper polynomial is a polynomial in at least one variable, or is the zero polynomial. An improper polynomial is a non-zero integer. Except where explicitly precluded, polynomial shall include both proper and improper polynomials. When an operation is to be performed on two polynomials, it is

frequently required that these polynomials be compatible. Two non-zero polynomials are compatible in case they are polynomials in the same variables and these variables occur in the same order in each (as a special case, any two improper polynomials are compatible). Also, the zero polynomial is compatible with any polynomial.

The compatibility requirements are not serious restrictions, since there are subprograms, described below in Section 3.5, which enable one to easily compatibilize any two polynomials by introducing vacuous occurrences of variables and changing the ordering of variables.

3.1 Input-Output

The following input-output conventions were established for the previous SAC-1 subsystems and continue to hold for the polynomial system. A standard record of 72 characters is used for input and output (either printed or punched). All input-output subprograms have an argument which specifies a logical unit number. In the case of output, the unit number determines, according to local convention, whether the output will be printed or punched. Each input or output item (e.g., polynomial) begins in column 1 of some record and extends over as many records as required, the last record being filled out with blanks. Blanks within an item are not admissible, except where explicit provision is made. In particular, no blanks are allowed within polynomials.

P = PREAD(U). U is a logical unit number. If U is positioned at the first record of a correctly punched polynomial, the records containing this polynomial are read, the polynomial is converted to internal canonical list representation, and P is the location of the resulting list. U is then left positioned at the next following record. If U was initially positioned at an end-of-file, P = -1 . If U was initially positioned at the first record of an incorrectly punched polynomial, P = -2 . However, not all errors will be detected. The last character of a polynomial should always be followed by a blank since this will terminate the attempt to read a polynomial in case of a missing parenthesis or other syntactical error. No diagnostics are printed.

PWRITE(U, P). U is a logical unit number, P is a polynomial. The polynomial P is converted to external canonical form and written as a sequence of records on unit U . (Note that the polynomial is not erased; if it is no longer needed, it should be erased using the PERASE subroutine.)

* PREADS(P, U, I). This is a subroutine called by PREAD which, in fact, does most of the work of the latter. U is a logical unit number, I is a FORTRAN integer, $1 \leq I \leq 72$, specifying a character position within a record. A polynomial is read from unit U , converted to internal list representation, and the location of this list is assigned to the variable P . It is assumed that the first record of the polynomial has already been read and is in the

array RECORD of common block TR1. Also, the first character of the polynomial is assumed to be in character position I of the first record. (When PREADS is called by PREAD, I will always be 1. But PREADS is designed for use by other subprograms in later SAC-1 subsystems also, in which case I need not be 1.) PREADS assigns a new value to I, namely, the character position in the last record which contained the last character of the polynomial. (If the polynomial is proper, the last character is a right parenthesis but if the polynomial is an integer, the last character is, by convention, a blank, which must be present, immediately following the last decimal digit.) As in PREAD, a value of -1 is assigned to P if U is at an end-of-file, -2 if a syntactic error is discovered.

* PWRITS(P, U, I). This subroutine bears the same relation to PWRITE as does PREADS to PREAD. P is a polynomial, U is a unit number, I is a character position. The polynomial P is converted to external canonical form and written as a sequence of records on unit U, starting in character position I of the first record. The first I-1 character positions of the first record will contain whatever characters were in the first I-1 positions of the array RECORD when PWRITS was called. The last record of the polynomial P is not written on unit U, but is left instead in RECORD. A new value is assigned to I, which is the character position in the last record which contains the last character of the polynomial (a right parenthesis if P is proper, a blank otherwise).

* I = PROSYM(X). PROSYM (PROcess SYMbol) manages the symbol list. X is the location of a list $(c_1, c_2, \ldots, c_n)$ where c_i is the SAC-1 internal code for the i-th character, C_i, of the symbol $S = C_1C_2\ldots C_n$. (A polynomial variable is an example of a symbol. Later SAC-1 subsystems may use symbols for other purposes, also, such as function names.) The infinite-precision integer J which represents S is generated and the symbol list is scanned for an occurrence of J. If J occurs on the symbol list, the list for J which was generated is erased and the value I is the location of the list for J which was on the symbol list.

If J is not on the symbol list, it is prefixed to the symbol list and the value I returned is the location of the list for J that was generated. If J occurred on the symbol list, the occurrence on the symbol list is borrowed. If J did not occur on the symbol list then the generated list for J is borrowed. This reference count policy for symbols results naturally from treating the symbol list just like any other list. Notice that it has the consequence that after all polynomials in which a variable occurs have been erased, that variable will still exist on the symbol list and have a reference count of one. Normally, the total number of variables will be sufficiently small that this will not create a problem. If it were to become a problem, however, one could always scan the symbol list, deleting and erasing those symbols with reference counts of one, and leaving all other symbols and their reference counts untouched.

* I = STOI(X). (Symbol TO Integer) X is the location of a list $(c_1, c_2, \ldots, c_n)$ where c_i is the SAC-1 internal code for the i-th character, C_i, of a symbol $C_1 C_2 \ldots C_n$. I is the infinite-precision integer $\Sigma_{i=1}^{n} c_i \cdot 64^{n-i}$.

* X = ITOS(I). (Integer TO Symbol) I is an infinite-precision integer $\Sigma_{i=1}^{n} c_i \cdot 64^{n-i}$ where c_i is the SAC-1 internal code for the i-th character, C_i, of a symbol $C_1 C_2 \ldots C_n$. X is the list $(c_1, c_2, \ldots, c_n)$.

* NEXTCH(U, I). (NEXT CHaracter) U is a logical unit number, I is a character position, $1 \leq I \leq 72$, within a record. If $I < 72$, I is replaced by $I + 1$. If $I = 72$, I is set to one and a new record is read from unit U into the array RECORD of common block TR1.

3.2. Arithmetic

R = PSUM(P, Q). P and Q are compatible polynomials. R = P + Q, a polynomial compatible with P and Q.

R = PNEG(P). P is an arbitrary polynomial. R = -P, a polynomial compatible with P.

R = PDIF(P, Q). P and Q are compatible polynomials. R = P - Q, a polynomial compatible with P and Q.

R = PPROD(P, Q). P and Q are compatible polynomials. R = P · Q, a polynomial compatible with P and Q.

R = PQ(P, Q). P and Q are compatible polynomials. R is the polynomial P/Q if $Q \neq 0$ and P/Q exists. Otherwise, R is the FORTRAN integer -1 .

R = PIP(P, I). (Polynomial-Integer Product) P is an arbitrary polynomial. I is an infinite-precision integer. R = P · I , a polynomial compatible with P .

R = PSPROD(P, Q, N). P and Q are polynomials and N is a non-negative FORTRAN integer. If either P or Q is zero, then R is zero. Otherwise, Q is compatible with the leading coefficient of Q , $R = P \cdot Q \cdot v^N$ where v is the main variable of P , and R is compatible with P .

R = PSQ(P, Q). (Polynomial Special Quotient) P is a non-zero proper polynomial. Q is a non-zero polynomial compatible with the leading coefficient of P . R = P/Q if this exists, and R is compatible with P . Otherwise, R = -1, a FORTRAN integer.

The following two subprograms require an explanation. They exist primarily to define uniquely the greatest common divisor of two polynomials. We define the leading numerical coefficient of a non-zero proper polynomial P recursively, as follows. If P is a polynomial in one variable, its leading numerical coefficient is its leading coefficient. Otherwise, the leading numerical coefficient is the leading numerical coefficient of the leading coefficient of P . Notice that the leading numerical coefficient of a poly-

nomial is the integer coefficient which occurs leftmost in its external canonical form. Notice also that the leading numerical coefficient is not invariant under a reordering of the variables.

The sign of a non-zero proper polynomial is defined to be the sign of its leading numerical coefficient. The sign of any other polynomial is its sign regarded as an integer. If P is any polynomial, the absolute value of P, $|P|$, is P if the sign of P is 0 or 1, -P otherwise. A positive polynomial is one whose sign is +1.

S = PSIGN(P). P is an arbitrary polynomial. S is the sign of P, a FORTRAN integer, +1, 0 or -1.

Q = PABS(P). P is an arbitrary polynomial. Q is the absolute value of P, a polynomial compatible with P.

R = PMPNV(P, V, N). P is a proper non-zero polynomial. V is a variable which occurs in P. N is a non-negative FORTRAN integer. R is the polynomial $P \cdot V^N$, a polynomial compatible with P.

3.3. Differentiation

R = PDERIV(P, V). P is an arbitrary polynomial. V is a variable (represented as an infinite-precision integer). R is the derivative of P with respect to V, a polynomial compatible with P. The variable V need not occur in P and, in particular, P may be an integer; in this case R is, of course, zero.

3.4. Greatest Common Divisors

R = PGCD(P, Q). P and Q are compatible polynomials. R is the greatest common divisor of P and Q , a polynomial compatible with P and Q . R is zero if P and Q are both zero; otherwise, R is a positive polynomial.

R = PCONT(P). P is a non-zero proper polynomial. R is the content of P , i.e., the greatest common divisor of the coefficients of P . R is a positive polynomial compatible with the leading coefficient of P .

R = PPP(P). P is a non-zero proper polynomial. R = $|P/Q|$ where Q is the content of P . R is compatible with P .

L = PCPP(P). P is a non-zero proper polynomial. L is the list (Q, R) where Q is the content of P and R is the primitive part of P .

R = PGCDA(P,Q). P and Q are non-zero proper compatible polynomials of respective degrees m and n in their main variable, with $m \geq n$. If $n = 0$, then R = Q. Otherwise, let $P_1, P_2, \ldots, P_k$ be the complete reduced polynomial remainder sequence (see [8] for a definition) starting with $P_1 = P$, $P_2 = Q$. If $P_k \neq 0$, then $R = P_k$; otherwise, $R = P_{k-1}$. R is an associate of the greatest common divisor of P and Q . R is compatible with P and Q .

R = PSREM(P, Q). P and Q are non-zero proper compatible polynomials of respective degrees m and n in their main variable, with $m \geq n$. R is the standardized remainder of P with respect to Q , a polynomial compatible

with P and Q . The standardized remainder can be defined as follows. Let v be the main variable of P and Q and let B be the leading coefficient of Q . Then R is the unique polynomial such that, for some polynomial S ,

(1) $B^{m-n+1} P = Q \cdot S + R$, and

(2) Either R = 0 or degree of R (in v) < degree of Q (in v).

* Z = PCGCD(I, P, Q). This subprogram exists as a means of implementing the method of indirect recursion described in [1] for the mutually recursive subprograms PCONT and PGCD .

3.5. Substitution and Reordering of Variables

R = PSUBST(P, Q). Either Q = 0, or else Q is a non-zero proper polynomial and P is a polynomial compatible with the leading coefficient of Q . R is the result of substituting P for the main variable of Q in Q , a polynomial compatible with the leading coefficient of Q (except R = 0 when Q = 0) .

This subprogram provides only for substitutions for the main variable. To make a substitution for some other variable of Q , one should reorder the variables of Q so that that variable becomes the main variable. In general, the variables of P must also be reordered so that P becomes compatible with the leading coefficient of the new Q . It may also be necessary to introduce vacuous occurrences of variables into either P or Q . This can all be done quite easily and efficiently using the following subprograms PORDER and PVLIST.

L = PVLIST(P). P is an arbitrary polynomial. If $P = P(x_1, \ldots, x_n)$, then L is the list of variables $(x_1, \ldots, x_n)$. If P is an integer, then L is the null list. Each variable in L is represented as an infinite-precision integer, as it is in P.

R = PORDER(P, L). L is a non-null list $(v_1, \ldots, v_n)$ of distinct variables such that every variable occurring in P occurs in L (L may also contain variables which do not occur in P). R is the polynomial $R(v_1, \ldots, v_n)$ which is equivalent to P and which results from P by reordering the variables (and, possibly, introducing others).

* R = PMERGE(P, Q). P and Q are non-zero, proper, compatible, disjoint polynomials (P and Q are <u>disjoint</u> in case they have no common terms, i.e. if $P = P(x_1, \ldots, x_n)$ and $Q = Q(x_1, \ldots, x_n)$, then the coefficient of $x_1^{e_1} \ldots x_n^{e_n}$ is zero in either P or Q for all n-tuples $(e_1, \ldots, e_n)$). R = P + Q, a polynomial compatible with P and Q. PMERGE is used in an important way by PORDER.

3.6. <u>Miscellaneous</u>

PERASE(P). P is an arbitrary polynomial. The list which represents P is "erased". The general subroutine ERASE would have the same effect, but PERASE executes faster because it takes advantage of the special structure of any list which represents a polynomial.

N = PDEG(P). P is a proper polynomial. N is the degree of P in its main variable, a FORTRAN integer (N = 0 if P = 0).

R = PLDCF(P). P is a proper polynomial. R is the leading coefficient of P (R = 0 if P = 0). The list which represents R is an element of the list for P ; the reference count of the former is increased by one.

V = PVBL(P). P is a proper polynomial. V is the main variable of P (V = 0 if P = 0). The list for V is an element of the list for P ; its reference count is increased by one.

R = PRED(P). P is a proper polynomial. R is the reductum of P . If P is the non-zero proper polynomial $\Sigma_{i=1}^{r} P_i(x_1, \ldots, x_n) \cdot y^{e_i}$, $e_1 > e_2 > \ldots > e_r$, then the reductum of P is $\Sigma_{i=2}^{r} P_i(x_1, \ldots, x_n) \cdot y^{e_i}$ (0 if r = 1); the reductum of 0 is 0.

S = PONE(P). P is an arbitrary polynomial. If $P(x_1, \ldots, x_n) = 1$ (for all $x_1, \ldots, x_n$), then S is the FORTRAN integer 1 ; otherwise S = 0. (Two polynomials, P and Q , are relatively prime just in case PONE(PGCD(P, Q)) = 1.)

4. Descriptions of the Algorithms

In this section we describe the algorithms embodied in the FORTRAN subprograms of the system. We have omitted these descriptions for the simplest subprograms, either because they are without interest or because one can quickly consult their listings in the appendix.

These verbal descriptions serve several purposes. In some cases they provide a guide to understanding the FORTRAN listing in the appendix. In some cases speical techniques which improve efficiency are explained. In some cases the mathematical formulas behind the algorithms are presented. One underlying theme is the simplicity of the algorithm which results from the use of the recursive canonical form for polynomials.

As in Section 3, the subprograms are grouped into six categories.

4.1. Input-Output

PREADS(P, U, I). This subroutine uses a recursive procedure, PREADS(Y), which reads and converts an arbitrary polynomial from external to internal form. If the polynomial is an integer (indicated by the absence of a leading left parenthesis), the integer is brought into the form of a list of characters and IDTOB is then applied and the list of characters is erased. If the polynomial is proper, its terms are processed in a loop. Each term consists of a coefficient, a variable, two asterisks and an exponent. The recursive procedure calls itself to process and convert a coefficient. Each occurrence of the variable is made into a list of characters. If the occurrence proves to

be not the last, it is simply erased; if it is the last, it is presented to PROSYM, which returns the location of the unique infinite-precision integer representation. The recursive procedure also calls itself to process each exponent; this yields an infinite-precision integer whose only element is the required FORTRAN integer. The recursive procedure returns the location of the list representation of the polynomial as the value of the variable Y, or -2 if a syntactic error is revealed.

I = STOI(X). This subprogram is essentially the same as IDTOB except that the conversion is from base 64 to base β rather than from base 10 to base β. The process consists in repeated multiplication by 64 and addition of the next character code. Each multiplication-addition is carried out by modifying the existing list.

X = ITOS(I). This is a base β to base 64 conversion. The inverse of the list for I is constructed with CINV. This inverse list is then repeatedly altered in carrying out successive divisions by 64.

4.2. Arithmetic

R = PSUM(P, Q). The trivial case where P or Q is zero is first disposed of. Then if P and Q are integers ISUM is used. Otherwise, recursion on the number of variables is used. The terms of P are merged with the terms of Q and the PSUM recursive procedure calls itself to add coefficients of like terms (except that ISUM is used directly if these coefficients are integers, as a means of improving efficiency). If the two

coefficients cancel, the corresponding term is omitted from the sum. When a term of P has no mate in Q (or vice versa), a coefficient of P is borrowed for use in R. When all terms of P (Q) have been processed, the remaining final segment of Q (P) is borrowed and made to be the final segment for R, after inverting the portion of the list for R that was constructed during the merge. The main variable of P is then borrowed and prefixed to the list for R. This borrowing of a final segment of P or Q, as an alternative to borrowing instead the coefficients in the final segment, surprisingly contributes a great deal to the efficiency of the algorithm. One would not expect such a final segment to be very long, on the average. However, PSUM is used as a subprogram of PPROD in such a way that there the expected length of this final segment is $\frac{1}{3}$ of the length of the entire list for R. This is one instance where the SAC-1 algorithm is an improvement of the corresponding PM algorithm.

R = PNEG(P). If P = 0, R = 0. If P is a non-zero integer, INEG is used. Otherwise, a recursive procedure is used, which calls itself to negate coefficients of P.

R = PDIF(P, Q). If Q = 0, R = P. If P = 0, PNEG is used. If P and Q are non-zero integers, IDIF is used. Otherwise, a recursive procedure is used, which parallels that in PSUM except that a final segment of P or Q is never borrowed. This algorithm is more efficient than using PNEG and PSUM because it avoids constructing (and later erasing) a list for -Q.

R = PPROD(P, Q). If P = 0 or Q = 0, then R = 0. If P and Q are non-zero integers, IPROD is used. Otherwise, a recursive procedure is entered. Inverses of the lists for P and Q , minus their variables, are constructed using CINV. From the inverse list for P and the first term of the inverse list for Q is constructed the product of P and the last (low order) term of Q . The recursive procedure calls itself once for each coefficient of this product. The list for the product is obtained in the correct order, without inversion, because the list for P was inverted. In like fashion, the product of P with each term of Q is constructed and is then added to the partial sum of all previous such products, using PSUM. The final partial sum is R .

The initial inversion of P obviates the inversion of the list for each of the partial products (products of P with terms of Q). The initial inversion of Q aids PSUM in computing the partial sums, since in each successive partial sum a longer final segment can be borrowed than in the previous sum. Both initial inversions represent improvements over the PM algorithm for polynomial sums.

R = PQ(P, Q). If Q = 0, R = -1. If P = 0 and $Q \neq 0$, R = 0. If P and Q are non-zero integers, IQR is used. If the degree of P is less than the degree of Q , R = -1 . Otherwise, the recursive procedure calls itself to attempt division of the leading coefficient of P by the leading coefficient of Q . If the attempt fails, then R = -1 . If it succeeds, the

quotient times the reductum of Q times a power of the main variable is subtracted from the reductum of P . The result replaces the polynomial P . If the new P is zero, P is divisible by Q and the coefficients of the quotient are the quotients of the coefficients that were formed. If the new P is non-zero, the process is repeated with the new P in place of the old.

R = PIP(P, I). If P = 0 or I = 0, R = 0. Otherwise, a recursive procedure is used. If P is an integer, IPROD is used. Otherwise, the recursive procedure calls itself to multiply each coefficient of P by I .

R = PMPNV(P, V, N). A recursive procedure is used. If V is the main variable, R is a replica of P in which each of the exponents is N larger than the corresponding exponent of P . If V is not the main variable, the recursive procedure uses itself to multiply each coefficient of P by V^N .

4.3. Differentiation

R = PDERIV(P, V). If P = 0, or if P is an integer, R = 0. Otherwise, a recursive procedure is used. If V is the main variable of P , each coefficient of P is multiplied by the exponent of the term in which it occurs, using PIP, and the exponent is reduced by one. If V is not the main variable, the recursive procedure uses itself to differentiate each coefficient of P with respect to V .

4.4. Greatest Common Divisors

The polynomial greatest common divisor algorithm employed in SAC-1 is the reduced polynomial remainder sequence algorithm described in [8]. The complete reduced polynomial remainder sequence $P_1, P_2, \ldots, P_k$ for two polynomials, P_1 and P_2, is computed by the subprogram PGCDA, which returns as its value the last non-zero element of this sequence, since this is an associated of the greatest common divisor of P_1 and P_2.

PGCD and PCONT are mutually recursive. As a result, in order to implement the method of indirect recursive procedures described in [1], there is a subprogram, PCGCD, which embodies recursive procedures for both PCONT and PGCD.

R = PGCD(P, Q). If P = 0, R = |Q|. If Q = 0, R = |P|. If P and Q are non-zero integers, IGCD is used. Otherwise, the recursive procedure for PGCD in the subprogram PCGCD is entered. A = PCONT(P) and B = PCONT(Q) are computed, then C = PGCD(A, B). Next $\overline{P}$ = P/A and $\overline{Q}$ = Q/B are computed. If the degree of $\overline{P}$ is less than the degree of $\overline{Q}$, $\overline{P}$ and $\overline{Q}$ are interchanged and then R = PGCDA($\overline{P}, \overline{Q}$). After that, D = PCONT(R) and $\overline{R}$ = R/D are computed. Finally, R = |R| · C is the greatest common divisor.

R = PCONT(P). The recursive procedure for PCONT in PCGCD is entered immediately. Let $P_1, P_2, \ldots, P_r$ be the coefficients of P. A is initialized to $|P_1|$. If r = 1, R = A. Otherwise, A is replaced by

PGCD(A, P_i), and PONE is applied to the new A, for i = 2,3, ... until an A is obtained which is identically one or until i = r, whichever occurs first. Then R = A.

R = PGCDA(P, Q). The reduced polynomial remainder sequence $P_1, P_2, P_3, P_4, \ldots$ is computed according to the following formulas. Let C_i be the leading coefficient of P_i, n_i the degree of P_i and $\delta_i = n_i - n_{i+1}$. Also, set $\delta_0 = -1$. Then $\overline{P}_{i+2}$ = PSREM(P_i, P_{i+1}) and $P_{i+2} = \overline{P}_{i+2}/C_i^{\delta_{i-1}+1}$. If $\overline{P}_{i+2} = 0$, then $R = P_{i+1}$. Otherwise, P_{i+2} is computed, P_i is replaced by P_{i+1}, P_{i+1} is replaced by P_{i+2}, and a new $\overline{P}_{i+2}$ is computed.

R = PSREM(P, Q). k = PDEG(P) - PDEG(Q) + 1 is computed. The following process is done k times. If PDEG(P) < PDEG(Q), then R = PLDCF(Q)·P. Otherwise, R = PLDCF(Q)·PRED(P) - PLDCF(P)·PRED(Q)·v^d where d = PDEG(P) - PDEG(Q) and v is the main variable. Then P is replaced by R before the process is repeated.

4.5. Substitution and Reordering of Variables

R = SUBST(P, Q). If Q = 0, then R = 0. Otherwise, Horner's method for the evaluation of a polynomial is used. Let $Q(x_1, \ldots, x_n, y) = \sum_{i=1}^{r} Q_i(x_1, \ldots, x_n) \cdot y^{e_i}$, $e_1 > e_2 > \ldots > e_r$. Starting with $R_0 = 0$, the recurrence formula $R_{i+1} = (R_i + Q_{i+1}) \cdot P^{(e_{i+1}-e_{i+2})}$ is used to compute $R_1, R_2, \ldots, R_r$, where by convention $e_{r+1} = 0$. Then $R = R_r$.

R = PORDER(P, L). If P = 0, R = 0. Otherwise, a recursive procedure is entered. If P is an integer, R is constructed from P by successively

introducing vacuous occurrences of the variables in the list L . If P is a proper polynomial, let $P = \sum_{i=1}^{r} P_i(x_1, \ldots, x_n) \cdot y^{e_i}$. Then $R = \sum_{i=1}^{r} P_i^* \cdot y^{e_i}$ where $P_i^* = \text{PORDER}(P_i, L)$. PORDER calls itself to compute each P_i^*. PMPNV computes each $P_i^* \cdot y^{e_i}$. The $P_i^* \cdot y^{e_i}$ are mutually disjoint. Hence PMERGE is used to compute the partial sums $\sum_{i=1}^{k} P_i^* \cdot y^{e_i}$.

R = PMERGE(P, Q). A recursive procedure is used. The terms of P are merged with the terms of Q . When a term of P matches a term of Q , the recursive procedure uses itself to add the two coefficients.

4.6. Miscellaneous

PERASE(P). A recursive procedure is entered unless P = 0. If P is an integer, ERLA is used. Otherwise, successive cells on the main level of P are returned to AVAIL by DECAP (until one is encountered whose reduced reference count is positive). The main variable of P is erased by ERLA (if at all). The recursive procedure calls itself to erase any coefficients of P which must be erased.

S = PONE(P). If P = 0, then S = 0 . If P is a proper polynomial, then S = 0 if deg (P) $\neq$ 0 and otherwise S = PONE(Q), where Q is the leading coefficient of P . If P is a non-zero integer, then S = 1 if and only if TAIL(P) = 0 and FIRST(P) = 1 .

5. A sample Application

As an illustration of the use of the SAC-1 polynomial system we include here a complete SAC-1 main program together with its input and output. This will serve to illustrate certain rules and practices which, if followed, will facilitate the programming of other applications problems.

The program below computes the terms, f_i and g_i, for $i = 1, 2, 3, \ldots$, of the f and g series of Keplerian motion. This calculation was first programmed for the FORMAC system [6] and the results were reported in [9]. For purposes of comparison, the calculation was later programmed for PM and the results were reported in [4].

Each f_i and each g_i is a polynomial, with integer coefficients, in three variables ϵ, μ and σ. ϵ, μ and σ are themselves functions of a time variable t, implicitly determined by the following differential equations in which a dot over a variable stands for its derivative with respect to t.

$$(1) \qquad \dot{\epsilon} = -\sigma(\mu + 2\epsilon), \quad \dot{\mu} = -3\mu\sigma, \quad \dot{\sigma} = \epsilon - 2\sigma^2 .$$

By the differential calculus, we have

$$(2) \qquad \dot{f}_i = \frac{\partial f_i}{\partial \epsilon} \cdot \dot{\epsilon} + \frac{\partial f_i}{\partial \mu} \cdot \dot{\mu} + \frac{\partial f_i}{\partial \sigma} \cdot \dot{\sigma},$$

with a like formula holding for $\dot{g}_i$. The f_i and g_i satisfy the following recurrence relations.

(3) $f_{i+1} = \dot{f}_i - \mu g_i , \quad g_{i+1} = f_i + \dot{g}_i .$

Finally, the initial conditions are

(4) $f_0 = 1 , \quad g_0 = 0 .$

In the program below, f_0, g_0, $\dot{\epsilon}$, $\dot{\mu}$ and $\dot{\sigma}$ are input polynomials, as is also μ . All polynomials are represented as polynomials in ϵ, μ and σ (in that order). In order to reduce the quantity of output, these variables are written as E, M and S . An integer N is also input, which terminates the calculation after f_N and g_N have been computed. In the sample output given, N = 12, although the program has been run for much larger N .

The program begins with several essential declarations. First, the labelled common blocks, TR1 and TR2, together with their constituent variables and arrays, are declared. Next, the array which is to be used as the available space list, here called SPACE, is dimensioned. Third, every variable, or array, and every function name which occurs in the program is declared to be of type integer. Omission of any variable or function name from the integer declarations is almost invariably disastrous and is one of the most common programming errors in using SAC-1.

This program was written to be run on a computer for which (under the current operating system) the standard input tape is logical unit 50 and the standard output tape is logical unit 51 . Hence the first two executable statements assign the integers 50 and 51 to IN and OUT , variables which are used throughout the program as arguments of PREAD and

PWRITE. In order to run this program on a different computer, it is then only necessary to change these two statement to assign different numbers to IN and OUT. Or, the program could be modified to produce punched output rather than printed output by merely assigning a different number to OUT.

Next, the BEGIN subroutine is called, which creates the available space list and initializes the pushdown stack. BEGIN must always be called before any other SAC-1 subprogram is referenced. Likewise, the symbol list must be initialized, by executing SYMLST = 0, before any polynomial subprogram is used.

Next, the input polynomials are read in. The input polynomials are listed after the program just as they were punched on the input cards. In this program the integer N was read in using IREAD. PREAD could have been used instead in the same way. N could also have been read using a FORTRAN read statement. Since it was read in as an infinite-precision integer, T , it must be converted to a FORTRAN integer. This is done using DECAP, which also serves to erase the list which represented N . the variables, ϵ, μ and σ, symbolically referenced as EPS, MU and SIGMA, are needed as arguments of PDERIV for computing the partial derivatives of the f_i and g_i. They are conveniently obtained using PVLIST and DECAP. Note well that the main variable, in this case SIGMA, is the <u>last</u> element of the list of variables provided by PVLIST.

After f_0 and g_0 have been printed at statement 40 and beyond, control returns to statement 20, where first $\dot{f}_i$ is computed (denoted by FDOT). Notice that each of the three derivatives, three products and two sums required for this computation must be written out as a separate FORTRAN statement in order that the intermediate polynomials created (FSIG, FMU, FEPS, P1, P2, P3 and T) can then be erased.

The next part of the program similarly computes $\dot{g}_i$, then f_{i+1} and g_{i+1}. Since f_{i+1} replaces f_i (as the value of F), f_{i+1} must be temporarily assigned to U, until the previous value of F can be erased.

This program was executed on a CDC 1604 computer in 75.9 seconds with on-line printing. A lengthy discussion of computing times and storage requirements is given in Section 6.

The FANDG program illustrates several programming requirements that are rather tedious and burdensome: the integer declarations of variables and function names, the inability to nest functions in most cases, and the necessity to individually and explicitly erase each polynomial or other list. It would not be difficult to design a preprocessor which would perform these chores automatically.

```
      PROGRAM FANDG
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      DIMENSION SPACE(15000)
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER PREAD,PVLIST,PDERIV,PPROD,PSUM,PDIF
      INTEGER SPACE,IN,OUT
      INTEGER F,G,MUPOL,MUDOT,SIGDOT,EPSDOT,FDOT,GDOT
      INTEGER VRLS,SIGMA,MU,EPS
      INTEGER FSIG,FMU,FEPS,GSIG,GMU,GEPS
      INTEGER P1,P2,P3,Q1,Q2,Q3,T,U,V
      IN=50
      OUT=51
      CALL BEGIN(SPACE,15000)
      SYMLST=0
      F=PREAD(IN)
      G=PREAD(IN)
      MUPOL=PREAD(IN)
      MUDOT=PREAD(IN)
      SIGDOT=PREAD(IN)
      EPSDOT=PREAD(IN)
      T=IREAD(IN)
      CALL DECAP(N,T)
      PRINT 14,N
14    FORMAT(/3H N=,I2)
      VRLS=PVLIST(F)
      CALL DECAP(EPS,VRLS)
      CALL DECAP(MU,VRLS)
      CALL DECAP(SIGMA,VRLS)
      I=0
      GO TO 40
20    FSIG=PDERIV(F,SIGMA)
      FMU=PDERIV(F,MU)
      FEPS=PDERIV(F,EPS)
      P1=PPROD(FSIG,SIGDOT)
      P2=PPROD(FMU,MUDOT)
      P3=PPROD(FEPS,EPSDOT)
      T=PSUM(P1,P2)
      FDOT=PSUM(T,P3)
      CALL PERASE(FSIG)
      CALL PERASE(FMU)
      CALL PERASE(FEPS)
      CALL PERASE(P1)
      CALL PERASE(P2)
      CALL PERASE(P3)
      CALL PERASE(T)
      GSIG=PDERIV(G,SIGMA)
      GMU=PDERIV(G,MU)
      GEPS=PDERIV(G,EPS)
      Q1=PPROD(GSIG,SIGDOT)
      Q2=PPROD(GMU,MUDOT)
      Q3=PPROD(GEPS,EPSDOT)
      T=PSUM(Q1,Q2)
      GDOT=PSUM(T,Q3)
```

```
      CALL PERASE(GSIG)
      CALL PERASE(GMU)
      CALL PERASE(GEPS)
      CALL PERASE(Q1)
      CALL PERASE(Q2)
      CALL PERASE(Q3)
      CALL PERASE(T)
      T=PPROD(G,MUPOL)
      U=PDIF(FDOT,T)
      V=PSUM(F,GDOT)
      CALL PERASE(F)
      CALL PERASE(G)
      CALL PERASE(T)
      CALL PERASE(FDOT)
      CALL PERASE(GDOT)
      F=U
      G=V
40    PRINT 41,I
41    FORMAT(/6H F SUB,I2)
      CALL PWRITE(OUT,F)
      PRINT 42,I
42    FORMAT(/6H G SUB,I2)
      CALL PWRITE(OUT,G)
      I=I+1
      IF (I-N) 20,20,21
21    STOP
      END
```

FANDG INPUT

```
(((+1E**0)M**0)S**0)
+0
(((+1E**0)M**1)S**0)
(((-2E**0)M**1)S**1)
(((-2E**0)M**0)S**2+((+1E**1)M**0)S**0)
(((-1E**0)M**1+(-2E**1)M**0)S**1)
+12
```

FANDG OUTPUT

```
N=12

F SUB 0
(((+1E**0)M**0)S**0)

G SUB 0
+0

F SUB 1
+0

G SUB 1
(((+1E**0)M**0)S**0)

F SUB 2
(((-1E**0)M**1)S**0)

G SUB 2
+0

F SUB 3
(((+2E**0)M**1)S**1)

G SUB 3
(((-1E**0)M**1)S**0)

F SUB 4
(((-15E**0)M**1)S**2+((+1E**0)M**2+(+2E**1)M**1)S**0)

G SUB 4
(((+6E**0)M**1)S**1)

F SUB 5
(((+105E**0)M**1)S**3+((-15E**0)M**2+(-45E**1)M**1)S**1)

G SUB 5
(((-45E**0)M**1)S**2+((+1E**0)M**2+(+9E**1)M**1)S**0)

F SUB 6
(((-945E**0)M**1)S**4+((+210E**0)M**2+(+630E**1)M**1)S**2+((-1E**0)M**3+
(-24E**1)M**2+(-45E**2)M**1)S**0)

G SUB 6
(((+420E**0)M**1)S**3+((-30E**0)M**2+(-180E**1)M**1)S**1)

F SUB 7
(((+10395E**0)M**1)S**5+((-3150E**0)M**2+(-9450E**1)M**1)S**3+((+63E**0)
M**2+(+882E**1)M**2+(+1575E**2)M**1)S**1)

G SUB 7
(((-4725E**0)M**1)S**4+((+630E**0)M**2+(+3150E**1)M**1)S**2+((-1E**0)M**
3+(-54E**1)M**2+(-225E**2)M**1)S**0)

F SUB 8
(((-135135E**0)M**1)S**6+((+51975E**0)M**2+(+155925E**1)M**1)S**4+((-220
5E**0)M**2+(-24570E**1)M**2+(-42525E**2)M**1)S**2+((+1E**0)M**4+(+117E**
1)M**3+(+1107E**2)M**2+(+1575E**3)M**1)S**0)
```

```
G SUB 8
(((+62370E**0)M**1)S**5+((-12600E**0)M**2+(-56700E**1)M**1)S**3+((+126E*
**0)M**2+(+2024E**1)M**2+(+9450E**2)M**1)S**1)

F SUB 9
(((+2227025E**0)M**1)S**7+((-945945E**0)M**2+(-2837835E**1)M**1)S**5+((+
65835E**0)M**3+(+644490E**1)M**2+(+1091475E**2)M**1)S**3+((-255E**0)M**4
 (-10935E**1)M**3+(-74385E**2)M**2+(-99225E**3)M**1)S**1)

G SUB 9
(((-945945E**0)M**1)S**6+((+250875E**0)M**2+(+1091475E**1)M**1)S**4+((-6
615E**0)M**3+(-111510E**1)M**2+(-297675E**2)M**1)S**2+((+1E**0)M**4+(+24
3E**1)M**3+(+4131E**2)M**2+(+11025E**3)M**1)S**0)

F SUB10
(((-34459425E**0)M**1)S**8+((+18918900E**0)M**2+(+56756700E**1)M**1)S**6
 ((-1891800E**0)M**3+(-17027010E**1)M**2+(-28378350E**2)M**1)S**4+((+211
20E**0)M**4+(+599940E**1)M**3+(+2421440E**2)M**2+(+4365900E**3)M**1)S**2
 ((-1E**0)M**5+(-408E**1)M**4+(-15066E**2)M**3+(-85410E**3)M**2+(-99225E
**4)M**1)S**0)

G SUB10
(((+16216200E**0)M**1)S**7+((-5675670E**0)M**2+(-22702680E**1)M**1)S**5+
((+263340E**0)M**3+(+2617460E**1)M**2+(+8721800E**2)M**1)S**3+((-510E**0
)M**4+(-35100E**1)M**3+(-371790E**2)M**2+(-793800E**3)M**1)S**1)

F SUB11
(((+6547290750E**0)M**1)S**9+((-41351313100E**0)M**2+(-1240539300E**1)M**1)
S**7+((+54864810E**0)M**3+(+465404940E**1)M**2+(+766215450E**2)M**1)S**5
 ((-1201200E**0)M**4+(-27027000E**1)M**3+(-137937700E**2)M**2+(-17027010
0E**3)M**1)S**3+((+1023E**0)M**5+(+114444E**1)M**4+(+2022758E**2)M**3+(+
9058500E**3)M**2+(+9823275E**4)M**1)S**1)

G SUB11
(((-3101348925E**0)M**1)S**8+((+1324323300E**0)M**2+(+510810300E**1)M**1)S
**6+((-945945E**0)M**3+(-11351313400E**1)M**2+(-255405150E**2)M**1)S**4+(
(+63360E**0)M**4+(+2589840E**1)M**3+(+21116700E**2)M**2+(+39293100E**3)M
**1)S**2+((-1E**0)M**5+(-1008E**1)M**4+(-50166E**2)M**3+(-457200E**3)M**
2+(-893025E**4)M**1)S**0)

F SUB12
(((-1274931057575E**0)M**1)S**10+((+98209361725E**0)M**2+(+29462808375E**1)
M**1)S**8+((-1640268630E**0)M**3+(-13315121820E**1)M**2+(-21709437750E**
2)M**1)S**6+((+581080950E**0)M**4+(+11229718500E**1)M**3+(+52986433350E**2)
M**2+(+63851287500E**3)M**1)S**4+((-195195E**0)M**5+(-1207260E**1)M**4+(
 15972975700E**2)M**3+(-61890183300E**3)M**2+(-63851287575E**4)M**1)S**2+((+1E
**0)M**6+(+2031E**1)M**5+(+164610E**2)M**4+(+2480958E**3)M**3+(+9951525E
**4)M**2+(+9823275E**5)M**1)S**0)

G SUB12
(((+6547290750E**0)M**1)S**9+((-3308104800E**0)M**2+(-1240539300E**1)M**
*1)S**7+((+329188860E**0)M**3+(+3587023440E**1)M**2+(+7662154500E**2)M**
1)S**5+((-480480E**0)M**4+(-14594580E**1)M**3+(-100540440E**2)M**2+(-
17027010E**3)M**1)S**3+((+2046E**0)M**5+(+255608E**1)M**4+(+9227196E**
2)M**3+(+60350400E**3)M**2+(+98232750E**4)M**1)S**1)
```

6. System Performance and Storage Requirements

The performance characteristics of SAC-1 will depend on many complex factors, including the type of problem to which it is applied, the computer on which it is implemented, and the FORTRAN compiler which is used to compile the SAC-1 subprograms. Nevertheless, we shall attempt below to draw some limited conclusions from experiments with the f and g series program above on the CDC 1604 computer and their comparison with PM and FORMAC performance on the same problem. In particular, we will present evidence that with only a little additional effort in implementing SAC-1, its performance can be brought to a level only slightly below that of PM .

In order to compare SAC-1 performance with PM performance, it is necessary to compare the CDC 1604 computer with the IBM 7094 computer. Although the ratio of speeds of the two computers will depend on the type of application, we believe that for the SAC-1 type of application, the IBM 7094 should be rated as 3.5 to 4.0 times as fast as the CDC 1604. The subprogram PFA is likely a rather accurate measure of this ratio. PM's PFA has an execution time of 44 microseconds. When programmed in assembly language, without subroutine calls, for the CDC 1604, execution time is 180 microseconds. This would indicate a ratio of 4.0 but other considerations (e.g., speed of arithmetic and I/O) suggest a somewhat smaller ratio.

The PM f and g series program, for N = 12, executed in 6.6 seconds (see [4]), as compared with 75.9 seconds for SAC-1. This is a ratio of 11.5, which reduces to about 3.0 after correction for change of computer. This implies that SAC-1 is about three times as slow as PM as a result of its computer independence. The explanation of this considerable loss is, of course, that the system spends a large percentage of its time just executing unproductive subroutine linkages. This state of affairs could have been avoided, had less pain been taken to make SAC-1 easy to implement by reducing the number of primitives to a minimum.

Fortunately, one can choose between ease of implementation and efficiency. One can begin with the standard version of SAC-1, in which only the primitives are programmed in assembly language. When occasion demands it, one can render the system more efficient by reprogramming in assembly language some of the simpler subprograms which were previously FORTRAN-compiled. In this reprogramming, the main objective is to program in-line certain simple list processing operations which were previously done via subroutine calls, thereby eliminating subroutine linkages.

The experiments we have performed indicate that it is possible to recoup a large portion of the lost efficiency with only a very small additional programming investment, since the source of the inefficiency is highly concentrated in a few of the lower-level subprograms which are frequently executed.

We reprogrammed in CDC 1604 assembly language six of the SAC-1 list processing subprograms: PFA, PFL, DECAP, ERLA, ADV and INV. Each was programmed so that it no longer contained any subroutine calls. The f and g series program, for N = 12, was then run three more times, with the following results.

Assembly Language Subprograms	Execution Time in Seconds
None	75.9
PFA, PFL	57.7
PFA, PFL, DECAP, ERLA	43.0
PFA, PFL, DECAP, ERLA, ADV, INV	38.5

Thus it appears that reprogramming these six simple subprograms, less than one man-month of effort, reduces the PM to SAC-1 efficiency ratio from 3.0 to 1.5, a very tolerable level. There is no guarantee that these ratios carry over to other computers and other SAC-1 applications, but we have reasons to believe that these figures will be representative.

The f and g series program, for N = 12, involves approximately 36000 executions of PFA and PFL . Prior to optimization, each of these subprograms had a CDC 1604 execution time of 680 microseconds. Thus these two subprograms alone accounted for 32% of the total execution time. Experiements with PM on other applications indicate that this percentage is typical. These experiments also showed that approximately

another one third of the total time is typically spent on list erasure. Since all polynomial erasure done by PERASE relies ultimately on DECAP and ERLA, it is not surprising that assembly language versions of these subprograms contributed appreciably to the above time reductions. The contribution of ADV and INV was much less substantial.

The amount of memory required for storage of the instructions and data (other than lists) of the SAC-1 subprograms may vary considerably from one computer to another. For whatever it is worth, in general, here are the approximate requirements for the CDC 1604. Two instructions are stored in most 1604 words, but the instruction repertoire is such that many instructions are frequently required to perform a simple task.

Subsystem	Memory Required
List Processing	1100
Integer Arithmetic	1300
Polynomial	2800

7. Acknowledgements

Development of the SAC-1 polynomial system has been supported by the University of Wisconsin Computing Center and, indirectly, by the Wisconsin Alumni Research Foundation and the Graduate School. Much of the programming was done by W. J. Fabens. Smaller portions were programmed by L. E. Heindel, M. T. McClellan and the author. The improved polynomial product algorithm was suggested in part by L. E. Heindel.

8. References

1. Collins, G. E. The SAC-1 List Processing System. University of Wisconsin Computing Center Report. July, 1967.

2. Collins, G. E. The SAC-1 Integer Arithmetic System. University of Wisconsin Computing Center Report. September, 1967.

3. Collins, G. E. PM, A System for Polynomial Manipulation. Comm. A.C.M. 9, 8 (Aug. 1966), 578-589.

4. Collins, G. E. and Griesmer, J. H. Comparison of Computing Times in ALPAK, FORMAC, PM and Korsvold's System. SICSAM Bulletin No. 4 (Sept. 1966), 20-25.

5. Brown, W. S., Hyde, J. P., and Tague, B. A. The ALPAK System for Non-numerical Algebra on a Digital Computer. Pt. I: Bell System Tech. J. 42, 5 (Sept. 1963), 2081-2119. Pt. II: Ibid. 43, 2 (March 1964), 785-804. Pt. III: Ibid. 43, 4 (July 1964), 1547-1562.

6. Bond, E. et.al. FORMAC -- An Experimental Formula Manipulation Compiler. Proceedings of the ACM National Conference, Aug. 1964.

7. McCarthy, John, et.al. LISP 1.5 Programmer's Manual. MIT Press, 1962.

8. Collins, G. E. Subresultants and Reduced Polynomial Remainder Sequences, Jour. A.C.M. 14, 1 (Jan. 1967), 128-142.

9. Sconzo, P., LeSchack, A. R., and Tobey, R. Symbolic Computation of f and g Series by Computer. Astronomical Journal, Vol. 70 (May 1965).

10. FORTRAN vs. Basic FORTRAN. Comm. A.C.M. 7, 10 (Oct. 1964), 592-625.

```
      INTEGER FUNCTION ITOS(X)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER X,Y,Z,Q,R,T
      INTEGER CINV,FIRST,TAIL,PFA
      Y=CINV(X)
      Z=0
1     Q=0
      T=Y
      R=FIRST(Y)
      IF(R.NE.0)GO TO 3
      CALL DECAP(R,Y)
      IF(Y.EQ.0)GO TO 4
      T=Y
2     R=FIRST(T)
3     CALL QR(Q,R,64)
      CALL ALTER(Q,T)
      Q=R
      T=TAIL(T)
      IF(T.NE.0)GO TO 2
      Z=PFA(Q,Z)
      GO TO 1
4     ITOS=Z
      RETURN
      END

      SUBROUTINE NEXTCH(U,I)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER U
      I=I+1
      IF(I.LE.72)RETURN
      I=1
      CALL READ(U,RECORD)
      RETURN
      END

      INTEGER FUNCTION PABS(X)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER X,P,S
      INTEGER PSIGN,PNEG,BORROW
      P=X
      S=PSIGN(P)
      IF(S.LT.0)PABS=PNEG(P)
      IF(S.GE.0)PABS=BORROW(P)
      RETURN
      END
```

```
      INTEGER FUNCTION PCGCD(I,X,Y)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER X,Y,P,Q,R,A,B,C,D,PB,QB,RB,RET
      INTEGER FIRST,TAIL,TYPE,PABS,PONE,PSQ,PSPROD,PGCDA,PDEG
      P=X
      Q=Y
      RET=1
      GO TO (10,20),I
1     PCGCD=R
      RETURN
C     RECURSIVE PROCEDURE R=PCONT(P)
10    P=TAIL(P)
      A=PABS(FIRST(P))
11    P=TAIL(TAIL(P))
      IF (P.EQ.0) GO TO 15
      B=FIRST(P)
      IF (TYPE(B).EQ.1) GO TO 12
      C=IGCD(A,B)
      GO TO 13
12    CALL STACK3(A,P,RET)
      P=A
      Q=B
      RET=2
      GO TO 20
14    C=R
      CALL UNSTK3(A,P,RET)
13    CALL PERASE(A)
      A=C
      IF (PONE(A).EQ.0) GO TO 11
15    R=A
      GO TO (1,14,21,22,23,24),RET
C     RECURSIVE PROCEDURE R=PGCD(P,Q)
20    IF (PDEG(P).GE.PDEG(Q)) GO TO 27
      A=P
      P=Q
      Q=A
27    CALL STACK3(P,Q,RET)
      RET=3
      GO TO 10
21    A=R
      CALL UNSTK3(P,Q,RET)
      CALL STACK2(P,Q)
      CALL STACK2(RET,A)
      P=Q
      RET=4
      GO TO 10
22    B=R
      CALL UNSTK2(RET,A)
      CALL UNSTK2(P,Q)
      IF(TYPE(A).EQ.0)GO TO 25
      CALL STACK3(P,Q,RET)
      CALL STACK2(A,B)
```

```
            P=A
            Q=B
            RET=5
            GO TO 20
25          C=IGCD(A,B)
            GO TO 26
23          C=R
            CALL UNSTK2(A,B)
            CALL UNSTK3(P,Q,RET)
26          PB=PSQ(P,A)
            QB=PSQ(Q,B)
            CALL PERASE(A)
            CALL PERASE(B)
            R=PGCDA(PB,QB)
            CALL PERASE(PB)
            CALL PERASE(QB)
            CALL STACK3(C,R,RET)
            P=R
            RET=6
            GO TO 10
24          D=R
            CALL UNSTK3(C,R,RET)
            RB=PSQ(R,D)
            CALL PERASE(R)
            CALL PERASE(D)
            R=PABS(RB)
            CALL PERASE(RB)
            RB=R
            R=PSPROD(RB,C,0)
            CALL PERASE(RB)
            CALL PERASE(C)
            GO TO (1,14,21,22,23,24),RET
            END

            INTEGER FUNCTION PCONT(X)
            COMMON /TR1/ AVAIL,STAK,RECORD(72)
            COMMON /TR2/ SYMLST
            INTEGER AVAIL,STAK,RECORD,SYMLST
            INTEGER PCGCD,X
            PCONT=PCGCD(1,X,0)
            RETURN
            END

            INTEGER FUNCTION PCPP(X)
            COMMON /TR1/ AVAIL,STAK,RECORD(72)
            COMMON /TR2/ SYMLST
            INTEGER AVAIL,STAK,RECORD,SYMLST
            INTEGER X,P,Q,R,A
            INTEGER PFL,PCONT,PSQ,PABS
            P=X
            A=PCONT(P)
            Q=PSQ(P,A)
            R=PABS(Q)
            CALL PERASE(Q)
            PCPP=PFL(A,PFL(R,0))
```

```
      RETURN
      END

      INTEGER FUNCTION PDEG(X)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER X,Y,Z
      INTEGER FIRST,TAIL
      Y=X
      Z=0
      IF(Y.NE.0) Z=FIRST(TAIL(TAIL(Y)))
      PDEG=Z
      RETURN
      END

      INTEGER FUNCTION PDERIV(X,Y)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER X,Y,P,Q,R,U,V,    C,D,E,F
      INTEGER BORROW,INV,PFA,PFL,PIP,TYPE
      P=X
      V=Y
      PDERIV=0
      IF (P.EQ.0) RETURN
      IF (TYPE(P).EQ.0) RETURN
      R=1
      GO TO 10
1     PDERIV=Q
      RETURN
10    Q=0
      IF (TYPE(P).EQ.0) GO TO (1,16),R
      CALL ADV(U,P)
      IF(V.NE.U)GO TO 13
14    CALL ADV(C,P)
      CALL ADV(E,P)
      IF (E.EQ.0) GO TO 15
      F=PFA(E,0)
      D=PIP(C,F)
      CALL ERLA(F)
      Q=PFL(D,Q)
      Q=PFA(E-1,Q)
15    IF (P.NE.0) GO TO 14
      GO TO 18
13    CALL ADV(C,P)
      CALL STACK2(P,Q)
      CALL STACK2(R,U)
      P=C
      R=2
      GO TO 10
16    D=Q
      CALL UNSTK2(R,U)
      CALL UNSTK2(P,Q)
```

```
          CALL ADV(E,P)
          IF (D.EQ.0) GO TO 17
          Q=PFL(D,Q)
          Q=PFA(E,Q)
17        IF (P.NE.0) GO TO 13
18        IF (Q.EQ.0) GO TO 19
          Q=INV(Q)
          Q=PFL(BORROW(U),Q)
19        GO TO (1,16),R
          END

          INTEGER FUNCTION PDIF(X,Y)
          COMMON /TR1/AVAIL,STAK,RECORD(72)
          COMMON /TR2/ SYMLST
          INTEGER AVAIL,STAK,RECORD,SYMLST
          INTEGER BORROW,TYPE,TAIL,PFA,PFL,INV,IDIF,PNEG
          INTEGER X,Y,P,Q,R,S,T,U,V,A,B,C,I,J,M,N
          P=X
          Q=Y
          IF (P.NE.0) GO TO 1
          PDIF=PNEG(Q)
          RETURN
1         IF (Q.NE.0) GO TO 2
          PDIF=BORROW(P)
          RETURN
2         IF (TYPE(P).NE.0) GO TO 3
          PDIF=IDIF(P,Q)
          RETURN
3         S=1
          GO TO 10
4         PDIF=R
          RETURN
10        R=0
          CALL ADV(I,P)
          Q=TAIL(Q)
11        U=P
          CALL ADV(A,P)
          CALL ADV(M,P)
12        V=Q
          CALL ADV(B,Q)
          CALL ADV(N,Q)
          GO TO 14
13        U=P
          CALL ADV(A,P)
          CALL ADV(M,P)
14        IF (M-N) 15,16,17
15        R=PFL(PNEG(B),R)
          R=PFA(N,R)
          IF (Q.NE.0) GO TO 12
          T=U
          J=1
          GO TO 25
17        R=PFL(BORROW(A),R)
          R=PFA(M,R)
          IF (P.NE.0) GO TO 13
```

```
            T=V
            J=-1
            GO TO 25
16          IF(TYPE(A).NE.0)GO TO 18
            C=IDIF(A,B)
            GO TO 19
18          CALL STACK3(P,Q,R)
            CALL STACK3(I,M,S)
            P=A
            Q=B
            S=2
            GO TO 10
20          C=R
            CALL UNSTK3(I,M,S)
            CALL UNSTK3(P,Q,R)
19          IF (C.EQ.0) GO TO 21
            R=PFL(C,R)
            R=PFA(M,R)
21          IF (P.NE.0) GO TO 22
            T=Q
            J=-1
            GO TO 25
22          IF (Q.NE.0) GO TO 11
            T=P
            J=1
25          IF (T.EQ.0) GO TO 26
            CALL ADV(A,T)
            IF (J.EQ.1) GO TO 27
            C=PNEG(A)
            GO TO 28
27          C=BORROW(A)
28          R=PFL(C,R)
            CALL ADV(M,T)
            R=PFA(M,R)
            GO TO 25
26          IF (R.EQ.0) GO TO 29
            R=PFL(BORROW(I),INV(R))
29          GO TO (4,20),S
            END

            SUBROUTINE PERASE(X)
            COMMON /TR1/AVAIL,STAK,RECORD(72)
            COMMON /TR2/ SYMLST
            INTEGER AVAIL,STAK,RECORD,SYMLST
            INTEGER TYPE,COUNT
            INTEGER X,P,R,V,A
            IF (X.EQ.0) RETURN
            P=X
            R=1
            GO TO 10
1           RETURN
10          IF (TYPE(P).EQ.1) GO TO 11
            CALL ERLA(P)
            GO TO (1,15),R
11          N=COUNT(P)-1
```

```
            IF (N.EQ.0) GO TO 12
            CALL SCOUNT(N,P)
            GO TO (1,15),R
12          CALL DECAP(V,P)
            CALL ERLA(V)
13          N=COUNT(P)-1
            IF (N.EQ.0) GO TO 14
            CALL SCOUNT(N,P)
            GO TO (1,15),R
14          CALL DECAP(A,P)
            CALL STACK2(P,R)
            P=A
            R=2
            GO TO 10
15          CALL UNSTK2(P,R)
            N=COUNT(P)-1
            IF (N.EQ.0) GO TO 16
            CALL SCOUNT(N,P)
            GO TO (1,15),R
16          CALL DECAP(M,P)
            IF (P.NE.0) GO TO 13
            GO TO (1,15),R
            END

            INTEGER FUNCTION PGCD(X,Y)
            COMMON /TR1/ AVAIL,STAK,RECORD(72)
            COMMON /TR2/ SYMLST
            INTEGER AVAIL,STAK,RECORD,SYMLST
            INTEGER X,Y,P,Q
            INTEGER TYPE,PABS,PCGCD
            P=X
            Q=Y
            IF (P.NE.0) GO TO 1
            PGCD=PABS(Q)
            RETURN
1           IF (Q.NE.0) GO TO 2
            PGCD=PABS(P)
            RETURN
2           IF (TYPE(P).NE.0) GO TO 3
            PGCD=IGCD(P,Q)
            RETURN
3           PGCD=PCGCD(2,P,Q)
            RETURN
            END

            INTEGER FUNCTION PGCDA(X,Y)
            COMMON /TR1/ AVAIL,STAK,RECORD(72)
            COMMON /TR2/ SYMLST
            INTEGER AVAIL,STAK,RECORD,SYMLST
            INTEGER C,CI,CJ,D,I,  NA,NB,P,Q,R,RA,X,Y
            INTEGER BORROW,PDEG,PLDCF,PPROD,PSREM,PSQ
            P=X
            Q=BORROW(Y)
            R=PSREM(P,Q)
```

```
      IF(R.EQ.0)GO TO 1
      NA=PDEG(P)
4     P=Q
      Q=R
      R=PSREM(P,Q)
      IF(R.EQ.0)GO TO 2
      NB=PDEG(P)
      D=NA-NB
      C=PLDCF(P)
      CALL PERASE(P)
      CI=BORROW(C)
      IF(D.EQ.0)GO TO 5
      DO 3 I=1,D
      CJ=PPROD(CI,C)
      CALL PERASE(CI)
3     CI=CJ
5     CALL PERASE(C)
      RA=PSQ(R,CI)
      CALL PERASE(R)
      CALL PERASE(CI)
      R=RA
      NA=NB
      GO TO 4
2     CALL PERASE(P)
1     PGCDA=Q
      RETURN
      END

      INTEGER FUNCTION PIP(X,Y)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER BORROW,INV,PFA,PFL,TYPE
      INTEGER X,Y,P,Q,R,S,V,A,B
      P=X
      Q=Y
      PIP=0
      IF(P.EQ.0.OR.Q.EQ.0)RETURN
      S=1
      GO TO 10
1     PIP=R
      RETURN
10    IF (TYPE(P).NE.0) GO TO 11
      R=IPROD(P,Q)
      GO TO (1,13),S
11    CALL ADV(V,P)
      R=0
12    CALL ADV(A,P)
      CALL STACK2(P,R)
      CALL STACK2(S,V)
      P=A
      S=2
      GO TO 10
13    B=R
      CALL UNSTK2(S,V)
```

```
      CALL UNSTK2(P,R)
      R=PFL(B,R)
      CALL ADV(A,P)
      R=PFA(A,R)
      IF (P.NE.0) GO TO 12
      R=INV(R)
      R=PFL(BORROW(V),R)
      GO TO (1,13),S
      END

      INTEGER FUNCTION PLDCF(X)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER X,Y,Z
      INTEGER BORROW,FIRST,TAIL
      Y=X
      Z=0
      IF(Y.NE.0) Z=BORROW(FIRST(TAIL(Y)))
      PLDCF=Z
      RETURN
      END

      INTEGER FUNCTION PMERGE(P,Q)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER A,B,F,G,I,M,MERGE,N,P,Q,R,RT,S,T,U,UP,V,VP,W
      INTEGER BORROW,INV,PFA,PFL
C     PROCEDURE CALL
      F=P
      G=Q
      R=1
      GO TO 24
25    PMERGE=MERGE
C     END PROCEDURE CALL
      RETURN
C     RECURSIVE PROCEDURE MERGE=PMERGE(F,G), RETURN TO R
24    S=F
      T=G
      MERGE=0
      CALL ADV(A,S)
      CALL ADV(B,T)
      MERGE=PFL(BORROW(A),MERGE)
7     I=1
4     M=S
      CALL ADV(U,S)
      CALL ADV(UP,S)
      GO TO (1,2),I
1     N=T
      CALL ADV(V,T)
      CALL ADV(VP,T)
2     IF(UP.EQ.VP) GO TO 6
      IF(UP.LT.VP) GO TO 5
```

```
      MERGE=PFL(BORROW(U),MERGE)
      MERGE=PFA(UP,MERGE)
      IF(S.NE.0) GO TO 3
      W=MERGE
      MERGE=INV(W)
      CALL SSUCC(BORROW(N),W)
      GO TO 20
3     I=2
      GO TO 4
5     MERGE=PFL(BORROW(V),MERGE)
      MERGE=PFA(VP,MERGE)
      IF(T.NE.0) GO TO 1
      W=MERGE
      MERGE=INV(W)
      CALL SSUCC(BORROW(M),W)
      GO TO 20
C     RECURSIVE PROCEDURE CALL RT=PMERGE(U,V)
6     CALL STACK3(MERGE,UP,R)
      CALL STACK2(S,T)
      F=U
      G=V
      R=2
      GO TO 24
26    RT=MERGE
      CALL UNSTK2(S,T)
      CALL UNSTK3(MERGE,UP,R)
C     END RECURSIVE PROCEDURE CALL
      MERGE=PFL(RT,MERGE)
      MERGE=PFA(UP,MERGE)
      IF(S.NE.0.AND.T.NE.0) GO TO 7
      W=MERGE
      MERGE=INV(W)
      IF(S.NE.0) GO TO 8
      CALL SSUCC(BORROW(T),W)
      GO TO 20
8     CALL SSUCC(BORROW(S),W)
C     PROCEDURE RETURN
20    GO TO (25,26),R
      END

      INTEGER FUNCTION PMPNV (P,V,E)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER A,B,E,FE,FP,FV,N,P,PPMPNV,R,RESULT,V,X
      INTEGER BORROW,INV,PFA,PFL
C     PROCEDURE CALL
      FP=P
      FV=V
      FE=E
      R=1
      GO TO 24
25    PMPNV=PPMPNV
C     END PROCEDURE CALL
      RETURN
```

```
C       RECURSIVE PROCEDURE PPMPNV=PMPNV(FP,FV,FE),RETURN TO R
24      A=FP
        X=FV
        N=FE
        PPMPNV=0
        CALL ADV(B,A)
        PPMPNV=PFL(BORROW(B),PPMPNV)
        IF(B.NE.X) GOTO 2
1       CALL ADV(B,A)
        PPMPNV=PFL(BORROW(B),PPMPNV)
        CALL ADV(B,A)
        B=B+N
        PPMPNV=PFA(B,PPMPNV)
        IF(A.NE.0) GO TO 1
        GO TO 20
2       CALL ADV(B,A)
C       RECURSIVE PROCEDURE CALL RESULT=PMPNV(B,X,N)
        CALL STACK3(A,PPMPNV,R)
        FP=B
        FV=X
        FE=N
        R=2
        GO TO 24
26      RESULT=PPMPNV
        CALL UNSTK3(A,PPMPNV,R)
C       END RECURSIVE PROCEDURE CALL
        PPMPNV=PFL(RESULT,PPMPNV)
        CALL ADV(B,A)
        PPMPNV=PFA(B,PPMPNV)
        IF(A.NE.0) GO TO 2
20      PPMPNV=INV(PPMPNV)
C       PROCEDURE RETURN
        GO TO (25,26),R
C       END RECURSIVE PROCEDURE, RETURN TO R
        END

        INTEGER FUNCTION PNEG(X)
        COMMON /TR1/AVAIL,STAK,RECORD(72)
        COMMON /TR2/ SYMLST
        INTEGER AVAIL,STAK,RECORD,SYMLST
        INTEGER BORROW,TYPE,INV,PFA,PFL,INEG
        INTEGER A,B,E,P,Q,R,X,V
        P=X
        IF (P.NE.0) GO TO 1
        PNEG=0
        RETURN
1       R=1
        GO TO 10
2       PNEG=Q
        RETURN
10      IF (TYPE(P).NE.0) GO TO 11
        Q=INEG(P)
        GO TO 14
11      Q=0
        CALL ADV(V,P)
```

```
12      CALL ADV(A,P)
        CALL STACK2(P,Q)
        CALL STACK2(R,V)
        P=A
        R=2
        GO TO 10
13      B=Q
        CALL UNSTK2(R,V)
        CALL UNSTK2(P,Q)
        Q=PFL(B,Q)
        CALL ADV(E,P)
        Q=PFA(E,Q)
        IF (P.NE.0) GO TO 12
        Q=PFL(BORROW(V),INV(Q))
14      GO TO (2,13),R
        END

        INTEGER FUNCTION PONE(X)
        COMMON /TR1/ AVAIL,STAK,RECORD(72)
        COMMON /TR2/ SYMLST
        INTEGER AVAIL,STAK,RECORD,SYMLST
        INTEGER     U,V,X
        INTEGER FIRST,TAIL,TYPE
        U=X
        IF(U.EQ.0)GO TO 1
3       IF(TYPE(U).EQ.0)GO TO 2
        V=TAIL(U)
        U=FIRST(V)
        IF(FIRST(TAIL(V)).EQ.0)GO TO 3
1       PONE=0
        RETURN
2       V=TAIL(U)
        IF(V.NE.0)GO TO 1
        U=FIRST(U)
        V=-1
        CALL ADD3(U,V,0)
        IF(V.NE.0)GO TO 1
        IF(U.NE.0)GO TO 1
        PONE=1
        RETURN
        END

        INTEGER FUNCTION PORDER(P,L)
        COMMON /TR1/ AVAIL,STAK,RECORD(72)
        COMMON /TR2/ SYMLST
        INTEGER AVAIL,STAK,RECORD,SYMLST
        INTEGER C,DUM,E,L,ORDER,P,PP,PPD,R,RET,R1,V,VL,VLD,V1,XPP,XVL
        INTEGER BORROW,PFA,PFL,PMERGE,PMPNV,TYPE
        PORDER=0
        IF(P.EQ.0) RETURN
C       PROCEDURE CALL
        XPP=P
        XVL=L
        RET=1
        GO TO 24
```

```
25      PORDER=ORDER
C       END PROCEDURE CALL
        RETURN
C       RECURSIVE PROCEDURE ORDER=PORDER(PP,VL),RETURN TO RET
24      PP=XPP
        VL=XVL
        ORDER=0
        IF(TYPE(PP).NE.0) GO TO 2
        VLD=VL
        PPD=BORROW(PP)
1       ORDER=PFA(0,0)
        ORDER=PFL(PPD,ORDER)
        CALL ADV(V,VLD)
        ORDER=PFL(BORROW(V),ORDER)
        IF(VLD.EQ.0) GO TO 20
        PPD=ORDER
        GO TO 1
2       CALL ADV(V1,PP)
5       CALL ADV(C,PP)
C       RECURSIVE PROCEDURE CALL
        CALL STACK2(ORDER,PP)
        CALL STACK2(V1,RET)
        XPP=C
        RET=2
        GO TO 24
26      R=ORDER
        CALL UNSTK2(V1,RET)
        CALL UNSTK2(ORDER,PP)
C       END RECURSIVE PROCEDURE CALL
        CALL ADV(E,PP)
        R1=PMPNV(R,V1,E)
        CALL PERASE(R)
        IF(ORDER.NE.0) GO TO 4
        ORDER=R1
        GO TO 6
4       DUM=ORDER
        ORDER=PMERGE(DUM,R1)
        CALL PERASE(DUM)
        CALL PERASE(R1)
6       IF(PP.NE.0) GO TO 5
C       PROCEDURE RETURN
20      GO TO (25,26),RET
C       END RECURSIVE PROCEDURE
        END

        INTEGER FUNCTION PPP(X)
        COMMON /TR1/ AVAIL,STAK,RECORD(72)
        COMMON /TR2/ SYMLST
        INTEGER AVAIL,STAK,RECORD,SYMLST
        INTEGER X,P,Q,A
        INTEGER PCONT,PSQ,PABS
        P=X
        A=PCONT(P)
        Q=PSQ(P,A)
        CALL PERASE(A)
```

```
      PPP=PABS(Q)
      CALL PERASE(Q)
      RETURN
      END

      INTEGER FUNCTION PQ(X,Y)
      INTEGER G,H,M,N,Q,R,S,T,U,V,X,XX,Y,YY,W
      INTEGER BORROW,FIRST,INV,IQR,PDEG,PDIF,PFA,PFL
      INTEGER PLDCF,PPROD,PRED,PVBL,TYPE,TAIL
      XX=X
      YY=Y
      Q=-1
      IF(YY.EQ.0) GO TO 99
      Q=0
      IF (XX.EQ.0) GO TO 99
      R=1
      GO TO 1
 99   PQ=Q
      RETURN
C BEGIN RECURSIVE PROCEDURE Q=PQ(XX,YY), RETURN LOCATION R.
 1    XX=BORROW(XX)
      IF(TYPE(XX).NE.0) GO TO 3
      M=IQR(XX,YY)
      CALL ERLA(XX)
      CALL DECAP(Q,M)
      CALL DECAP(S,M)
      IF(S.EQ.0) GO TO 11
      CALL ERLA(Q)
      CALL ERLA(S)
      Q=-1
      GO TO 11
 3    H=PDEG(YY)
      U=PRED(YY)
      Q=0
 4    V=PVBL(YY)
      G=PDEG(XX)-H
      IF(G.LT.0) GO TO 7
C RECURSIVE CALL USING LEADING COEFFICIENTS OF XX AND YY AS ARGUMENTS.
      CALL STACK3(XX,YY,Q)
      CALL STACK3(U,V,R)
      CALL STACK2(H,G)
      XX=FIRST(TAIL(XX))
      YY=PLDCF(YY)
      R=2
      GO TO 1
 5    M=Q
      CALL PERASE(YY)
      CALL UNSTK2(H,G)
      CALL UNSTK3(U,V,R)
      CALL UNSTK3(XX,YY,Q)
C END RECURSIVE CALL WITH QUOTIENT M AND DEGREE G.
      IF(M.EQ.-1) GO TO 7
      N=PFL(V,PFL(BORROW(M),PFA(G,0)))
      T=PPROD(N,U)
```

```
      S=PRED(XX)
      W=PDIF(S,T)
      CALL PERASE(N)
      CALL PERASE(T)
      CALL PERASE(S)
      CALL PERASE(XX)
      XX=W
      Q=PFA(G,PFL(M,Q))
      IF(XX.NE.0) GO TO 4
      Q=PFL(BORROW(V),INV(Q))
      GO TO 10
 7    CALL PERASE(XX)
      IF(Q.NE.0) GO TO 8
      CALL ERLA(V)
      GO TO 9
 8    CALL PERASE(PFL(V,INV(Q)))
 9    Q=-1
 10   CALL PERASE(U)
 11   GO TO (99,5),R
C END RECURSIVE PROCEDURE PQ WITH QUOTIENT Q.
      END

      INTEGER FUNCTION PPROD(X,Y)
      INTEGER FIRST,TAIL,BORROW,CINV,TYPE
      INTEGER PFA,PFL,PSUM
      INTEGER X,Y,P,Q,R,S,V,PI,QI,PT,T,A,B,C,U
      IF (X.NE.0.AND.Y.NE.0) GO TO 1
      PPROD=0
      RETURN
1     IF (TYPE(X).NE.0) GO TO 2
      PPROD=IPROD(X,Y)
      RETURN
2     P=X
      Q=Y
      S=1
      GO TO 10
3     PPROD=R
      RETURN
10    R=0
      V=FIRST(P)
      PI=CINV(TAIL(P))
      QI=CINV(TAIL(Q))
11    T=0
      CALL DECAP(N,QI)
      CALL DECAP(B,QI)
      PT=PI
12    CALL ADV(M,PT)
      CALL ADV(A,PT)
      IF (TYPE(A).EQ.1) GO TO 13
      C=IPROD(A,B)
      GO TO 15
13    CALL STACK3(PI,QI,PT)
      CALL STACK3(V,M,N)
      CALL STACK2(R,S)
```

```
      CALL STACK2(T,B)
      P=A
      Q=B
      S=2
      GO TO 10
14    C=R
      CALL UNSTK2(T,B)
      CALL UNSTK2(R,S)
      CALL UNSTK3(V,M,N)
      CALL UNSTK3(PI,QI,PT)
15    T=PFA(M+N,T)
      T=PFL(C,T)
      IF (PT.NE.0) GO TO 12
      T=PFL(BORROW(V),T)
      CALL PERASE(B)
      U=PSUM(R,T)
      CALL PERASE(R)
      CALL PERASE(T)
      R=U
      IF (QI.NE.0) GO TO 11
16    CALL DECAP(M,PI)
      CALL DECAP(A,PI)
      CALL PERASE(A)
      IF (PI.NE.0) GO TO 16
      GO TO (3,14),S
      END

      INTEGER FUNCTION PREAD(U)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      INTEGER AVAIL,STAK,RECORD
      INTEGER I,X,U
      CALL READ(U,RECORD)
      I=1
      IF(RECORD(1).EQ.-1)GO TO 1
      CALL PREADS(X,U,I)
      PREAD=X
      RETURN
1     PREAD=-1
      RETURN
      END

      SUBROUTINE PREADS(X,U,I)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      INTEGER AVAIL,STAK,RECORD
      INTEGER X,U,I,Y,V,J,R,W,L,C,E,N
      INTEGER PFA,INV,IDTOB,PFL,PROSYM,FIRST
      V=U
      J=I
      R=1
      GO TO 10
1     X=Y
      I=J
      RETURN
C     RECURSIVE PROCEDURE PREADS(Y), RETURN VARIABLE R
```

```
10      Y=0
        IF(RECORD(J).EQ.40)GO TO 20
C       READ INTEGER
        W=0
        L=RECORD(J)
        IF(L.EQ.36.OR.L.EQ.37)GO TO 11
        IF(L.LT.0.OR.L.GT.9)GO TO 50
        GO TO 11
12      IF(RECORD(J).LT.0.OR.RECORD(J).GT.9)GO TO 13
11      W=PFA(RECORD(J),W)
        CALL NEXTCH(V,J)
        GO TO 12
13      W=INV(W)
        Y=IDTOB(W)
        CALL ERLA(W)
        GO TO (1,21,25),R
C       READ PROPER POLYNOMIAL
C       READ COEFFICIENT
20      CALL NEXTCH(V,J)
27      CALL STACK2(Y,R)
        R=2
        GO TO 10
21      C=Y
        CALL UNSTK2(Y,R)
        IF(C.LT.0)GO TO 51
        Y=PFL(C,Y)
        IF(RECORD(J).EQ.41)CALL NEXTCH(V,J)
C       READ VARIABLE
        IF(RECORD(J).LT.10.OR.RECORD(J).GT.35)GO TO 51
        W=0
        GO TO 23
22      CALL NEXTCH(V,J)
        IF(RECORD(J).LT.0.OR.RECORD(J).GT.35)GO TO 24
23      W=PFA(RECORD(J),W)
        GO TO 22
C       READ TWO ASTERISKS
24      IF(RECORD(J).NE.38)GO TO 50
        CALL NEXTCH(V,J)
        IF(RECORD(J).NE.38)GO TO 50
        CALL NEXTCH(V,J)
C       READ EXPONENT
        IF(RECORD(J).EQ.37)GO TO 50
        CALL STACK3(W,Y,R)
        R=3
        GO TO 10
25      E=Y
        CALL UNSTK3(W,Y,R)
        N=0
        IF(E.EQ.0)GO TO 28
        N=FIRST(E)
        CALL ERLA(E)
28      Y=PFA(N,Y)
C       ARE THERE MORE TERMS
        IF(RECORD(J).EQ.41)GO TO 30
```

```
      IF(RECORD(J).EQ.37)GO TO 26
      IF(RECORD(J).NE.36)GO TO 50
      CALL NEXTCH(V,J)
26    CALL ERLA(W)
      GO TO 27
C     NO MORE TERMS
30    W=INV(W)
      E=PROSYM(W)
      CALL ERLA(W)
      Y=PFL(E,INV(Y))
      GO TO(1,21,25),R
C     ERROR RETURN
50    CALL ERLA(W)
51    CALL ERASE(Y)
      Y=-2
      GO TO (1,21,25),R
      END

      INTEGER FUNCTION PRED(X)
      INTEGER W,X,Y,Z
      INTEGER BORROW,FIRST,PFL,TAIL
      Y=X
      Z=0
      IF(Y.EQ.0) GO TO 1
      W=TAIL(TAIL(TAIL(Y)))
      IF(W.EQ.0) GO TO 1
      Z=PFL(BORROW(FIRST(Y)),BORROW(W))
 1    PRED=Z
      RETURN
      END

      INTEGER FUNCTION PROSYM(X)
      COMMON /TR2/ SYMLST
      INTEGER SYMLST
      INTEGER A,B,D,L,X
      INTEGER STOI,IDIF,BORROW,PFL
      A=STOI(X)
      L=SYMLST
1     IF(L.EQ.0)GO TO 2
      CALL ADV(B,L)
      D=IDIF(A,B)
      IF(D.EQ.0)GO TO 3
      CALL ERLA(D)
      GO TO 1
2     SYMLST=PFL(BORROW(A),SYMLST)
      PROSYM=A
      RETURN
3     CALL ERLA(A)
      PROSYM=BORROW(B)
      RETURN
      END
```

```
      INTEGER FUNCTION PSIGN(X)
      INTEGER P,X
      INTEGER FIRST,ISIGNL,TAIL,TYPE
      P=X
      IF(P.EQ.0)GO TO 1
3     IF(TYPE(P).EQ.0)GO TO 2
      P=FIRST(TAIL(P))
      GO TO 3
2     PSIGN=ISIGNL(P)
      RETURN
1     PSIGN=0
      RETURN
      END

      INTEGER FUNCTION PSPROD(X,Y,N)
      INTEGER P,Q,R,X,Y
      INTEGER BORROW,PFA,PFL,PPROD,PVBL
      P=X
      Q=Y
      PSPROD=0
      IF(P.EQ.0.OR.Q.EQ.0)RETURN
      R=PFL(PVBL(P),PFL(BORROW(Q),PFA(N,0)))
      PSPROD=PPROD(P,R)
      CALL PERASE(R)
      RETURN
      END

      INTEGER FUNCTION PSQ(X,Y)
      INTEGER P,Q,R,X,Y
      INTEGER BORROW,FIRST,PFA,PFL,PQ
      P=X
      Q=Y
      R=PFL(BORROW(FIRST(P)),PFL(BORROW(Q),PFA(0,0)))
      PSQ=PQ(P,R)
      CALL PERASE(R)
      RETURN
      END

      INTEGER FUNCTION PSREM(X,Y)
      INTEGER X,Y,P,Q,N,K,QL,QR,I,J,PL,PR,A,B
      INTEGER BORROW,PDEG,PLDCF,PRED,PSPROD,PDIF
      P=BORROW(X)
      Q=Y
      N=PDEG(Q)
      K=PDEG(P)-N+1
      QL=PLDCF(Q)
      QR=PRED(Q)
      DO 2 I=1,K
      IF(P.EQ.0)GO TO 3
      J=PDEG(P)-N
      IF(J.LT.0)GO TO 1
      PL=PLDCF(P)
      PR=PRED(P)
      A=PSPROD(PR,QL,0)
```

```
      B=PSPROD(QR,PL,J)
      CALL PERASE(P)
      CALL PERASE(PL)
      CALL PERASE(PR)
      P=PDIF(A,B)
      CALL PERASE(A)
      CALL PERASE(B)
      GO TO 2
1     A=PSPROD(P,QL,0)
      CALL PERASE(P)
      P=A
2     CONTINUE
3     CALL PERASE(QL)
      CALL PERASE(QR)
      PSREM=P
      RETURN
      END

      INTEGER FUNCTION PSUBST(P,Q)
      INTEGER DUMP,EJ,FLAG,I,IP,P,PX,Q,QI,QJ,QX
      INTEGER PERASE,PPROD,PSUM
      PX=P
      QX=Q
      PSUBST=0
      IF(QX.EQ.0) RETURN
      FLAG=0
      CALL ADV(DUMP,QX)
      CALL ADV(QI,QX)
      CALL ADV(IP,QX)
6     IF(QX.NE.0) GO TO 1
      FLAG=1
      GO TO 2
1     CALL ADV(QJ,QX)
      CALL ADV(EJ,QX)
      IP=IP-EJ
2     DUMP=PSUBST
      PSUBST=PSUM(DUMP,QI)
      CALL PERASE(DUMP)
      IF(IP.EQ.0) RETURN
      DO 3 I=1,IP
      DUMP=PSUBST
      PSUBST=PPROD(DUMP,PX)
3     CALL PERASE(DUMP)
4     IF(FLAG.EQ.1) RETURN
      QI=QJ
      IP=EJ
      GO TO 6
      END
```

```
      INTEGER FUNCTION PSUM(X,Y)
      INTEGER BORROW,TYPE,TAIL,PFA,PFL,INV,ISUM
      INTEGER X,Y,P,Q,R,S,T,U,V,W,A,B,C,I,M,N
      P=X
      Q=Y
      IF (P.NE.0) GO TO 1
      PSUM=BORROW(Q)
      RETURN
1     IF (Q.NE.0) GO TO 2
      PSUM=BORROW(P)
      RETURN
2     IF (TYPE(P).NE.0) GO TO 3
      PSUM=ISUM(P,Q)
      RETURN
3     S=1
      GO TO 10
4     PSUM=R
      RETURN
10    R=0
      CALL ADV(I,P)
      Q=TAIL(Q)
11    U=P
      CALL ADV(A,P)
      CALL ADV(M,P)
12    V=Q
      CALL ADV(B,Q)
      CALL ADV(N,Q)
      GO TO 14
13    U=P
      CALL ADV(A,P)
      CALL ADV(M,P)
14    IF (M-N) 15,16,17
15    R=PFL(BORROW(B),R)
      R=PFA(N,R)
      IF (Q.NE.0) GO TO 12
      T=U
      GO TO 25
17    R=PFL(BORROW(A),R)
      R=PFA(M,R)
      IF (P.NE.0) GO TO 13
      T=V
      GO TO 25
16    IF (TYPE(A).NE.0) GO TO 18
      C=ISUM(A,B)
      GO TO 19
18    CALL STACK3(P,Q,R)
      CALL STACK3(I,M,S)
      P=A
      Q=B
      S=2
      GO TO 10
20    C=R
      CALL UNSTK3(I,M,S)
      CALL UNSTK3(P,Q,R)
```

```
19      IF (C.EQ.0) GO TO 21
        P=PFL(C,R)
        R=PFA(M,R)
21      IF(P.NE.0)GO TO 22
        T=Q
        GO TO 25
22      IF (Q.NE.0) GO TO 11
        T=P
25      IF (R.NE.0) GO TO 26
        IF(T.NE.0)R=PFL(BORROW(I),BORROW(T))
        GO TO (4,20),S
26      W=PFL(BORROW(I),INV(R))
        CALL SSUCC(BORROW(T),R)
        R=W
        GO TO (4,20),S
        END

        INTEGER FUNCTION PVBL(X)
        INTEGER X,Y,Z
        INTEGER BORROW,FIRST
        Y=X
        Z=0
        IF(Y.NE.0) Z=BORROW(FIRST(Y))
        PVBL=Z
        RETURN
        END

        INTEGER FUNCTION PVLIST(P)
        INTEGER A,B,P
        INTEGER BORROW,PFL,TYPE
        A=P
        PVLIST=0
        IF(A.EQ.0) RETURN
1       IF(TYPE(A).EQ.0) RETURN
        CALL ADV (B,A)
        PVLIST=PFL(BORROW(B),PVLIST)
        CALL ADV(B,A)
        A=B
        GO TO 1
        END

        SUBROUTINE PWRITE(U,P)
        COMMON /TR1/ AVAIL,STAK,RECORD(72)
        INTEGER AVAIL,STAK,RECORD
        INTEGER P,U,I,J
        I=1
        CALL PWRITS(P,U,I)
        IF(I.EQ.72)GO TO 1
        DO 2 J=I,71
2       RECORD(J+1)=44
1       CALL WRITE(U,RECORD)
        RETURN
        END
```

```
      SUBROUTINE PWRITS(X,U,I)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      INTEGER AVAIL,STAK,RECORD
      INTEGER X,U,I
      INTEGER P,R,A,V,S,L,Q,VR,E,T
      INTEGER TYPE,PFA,IBTOD,ITOS
      P=X
      IF(P.EQ.0)GO TO 100
      IF(TYPE(P).EQ.1)GO TO 200
100   ASSIGN 110 TO Q
      A=IBTOD(P)
      CALL DECAP(L,A)
      GO TO 2
110   IF(A.EQ.0)GO TO 120
      CALL DECAP(L,A)
      GO TO 1
120   L=44
      ASSIGN 999 TO Q
      GO TO 1
200   ASSIGN 210 TO Q
      L=40
      A=0
      R=1
      GO TO 2
210   CALL ADV(V,P)
      V=ITOS(V)
      CALL ADV(T,P)
      IF(TYPE(T).EQ.0)GO TO 300
220   CALL STACK3(V,P,R)
      P=T
      R=2
      L=40
      ASSIGN 210 TO Q
      GO TO 1
300   S=IBTOD(T)
      ASSIGN 310 TO Q
320   CALL DECAP(L,S)
      GO TO 1
310   IF(S.NE.0)GO TO 320
400   VR=V
      ASSIGN 410 TO Q
420   CALL ADV(L,VR)
      GO TO 1
410   IF(VR.NE.0)GO TO 420
      L=38
      ASSIGN 500 TO Q
      GO TO 1
500   ASSIGN 600 TO Q
      GO TO 1
600   CALL ADV(E,P)
      ASSIGN 610 TO Q
620   L=E/10
      E=E-L*10
      A=PFA(E,A)
```

```
      E=L
      IF(E.NE.0)GO TO 620
630   CALL DECAP(L,A)
      GO TO 1
610   IF(A.NE.0)GO TO 630
      L=41
      ASSIGN 700 TO Q
      IF(P.EQ.0)GO TO 1
      CALL ADV(T,P)
      IF(TYPE(T).EQ.0)GO TO 300
      ASSIGN 220 TO Q
      L=36
      GO TO 1
700   CALL ERLA(V)
      GO TO(999,710),R
710   CALL UNSTK3(V,P,R)
      GO TO 400
999   RETURN
1     I=I+1
      IF(I.LE.72)GO TO 2
      I=1
      CALL WRITE(U,RECORD)
2     RECORD(I)=L
      GO TO Q,(110,210,220,310,410,500,600,610,700,999)
      END


      INTEGER FUNCTION STOI(X)
      INTEGER S,Y,X,E,B,R,T
      INTEGER PFA,FIRST,TAIL
      Y=X
      CALL ADV(E,Y)
      S=PFA(E,0)
1     IF(Y.EQ.0)GO TO 3
      CALL ADV(E,Y)
      B=E
      R=S
2     E=FIRST(S)
      T=64
      CALL MPY(E,T)
      CALL ADD3(B,T,0)
      B=B+E
      CALL ALTER(T,S)
      T=S
      S=TAIL(S)
      IF(S.NE.0)GO TO 2
      IF (B.NE.0) CALL SSUCC(PFA(B,0),T)
      S=R
      GO TO 1
3     STOI=S
      RETURN
      END
```