THE SAC-1 MODULAR ARITHMETIC SYSTEM

by

G. E. Collins, L. E. Heindel, E. Horowitz,
M. T. McClellan and D. R. Musser

Technical Report #165

November 1972

Abstract

This is a reprinting of the original report of June 1969, with correction of a few minor errors. The SAC-1 Modular Arithmetic System is the fifth of the ten SAC-1 subsystems which are now available. It provides subprograms for the arithmetic operations in a prime finite field GF(p) , for any single-precision prime p , and for various operations on polynomials in several variables with coefficients in GF(p) . Besides the arithmetic operations on such polynomials there are included subprograms for the Chinese remainder theorem, evaluation and interpolation. For univariate polynomials, subprograms are included for greatest common divisor calculation and Berlekamp's factorization algorithm.

Table of Contents

1. Introduction

The SAC-1 Modular Arithmetic System is the fifth subsystem of the SAC-1 System for Symbolic and Algebraic Calculation. The four previously completed subsystems are the List Processing System [2], the Revised Infinite Precision Integer Arithmetic System [4], the Polynomial System [5], and the Rational Function System [6]. An original Infinite Precision Integer Arithmetic System [3] has been superceded by the revised system [4], which provides some additional operations, utilizes improved algorithms, and requires fewer primitive (machine language) subprograms.

The Modular Arithmetic System provides a common basis for several SAC-1 subsystems which are currently in various stages of development. A Revised Polynomial System is scheduled for release in August, 1969. This system will feature a modular arithmetic algorithm for multivariate polynomial g.c.d. calculation (along the lines suggested in [10], pages 393-395), which is several orders of magnitude faster than the reduced p.r.s. g.c.d. algorithm used in the existing system. The Revised Polynomial System will also provide some new operations, e.g., resultant calculation.

A Revised Rational Function System and a Rational Function Integration System are expected to be ready for release about October, 1969. Distribution of a Polynomial Zero Calculation System is tentatively scheduled for January, 1970, and a Linear Algebra System will be completed about the same time. A Polynomial Factorization System which is in an early stage of development may be available by late 1970.

In the SAC-1 Modular Arithmetic System, the moduli are assumed to be single-precision prime numbers. No attempt has been made to provide a general modular arithmetic system, the use of single-precision moduli being optimal for all the intended applications referred to above. Thus, the system performs arithmetic operations in any field GF(p), if p is single-precision. This, of course, is quite trivial. Beyond this, the system performs a wide variety of operations on univariate and multivariate polynomials over GF(p). These operations are categorized in Section 3, where the programs and algorithms for the operations are described. The system also includes a program for the rapid generation of prime numbers to be used in the system as moduli and, of course, programs pertaining to the Chinese remainder theorem.

Section 2 specifies the manner in which data objects are represented in the system. All the system programs are written in A.S.A. FORTRAN, [1], and these programs are listed (in alphabetical order) in Section 5. All programs have been extensively tested, but if any error is discovered, however, minor, the first-listed author would appreciate notification. Readers and users are assumed to be already familiar with the previous SAC-1 system reports cited above, of which copies are available upon request.

The SAC-1 System is easy to implement on most computers with FORTRAN compilers since only a few very simple primitive subprograms need be programmed in machine language. SAC-1 has been implemented on numerous computers and inquiries about implementation are encouraged.

In the present manual, we give a theoretical maximum computing time for each subprogram, expressed in O-notation as a function of parameters describing the inputs. A supplement to this manual is planned, which will approximate average computing times in terms of the same parameters. These approximation formulas will be derived from empirically observed computing times for a particular computer, the UNIVAC 1108, but they should also provide at least a guide to computing times for other computers through the application of an appropriate correction factor.

2. Data Specification

In the Modular Arithmetic System, the prime moduli, p, are assumed to satisfy $p<\beta/2$, where β is the radix used in the Revised Infinite Precision Integer Arithmetic System [4]. In the Chinese remainder theorem programs, the moduli are required to be odd, hence it is generally best to avoid using $p=2$. In any case, it is generally best to use very large primes, say $p>\beta/4$, the exception being in some of the polynomial factorization programs, in which only relatively small primes are feasible. The elements of GF(p) are taken to be the integers $0,1,2,\ldots,p-1$, which, along with the modulus p, are represented as FORTRAN integers.

Polynomials over GF(p) are represented in a recursive canonical form similar to that used in the SAC-1 Polynomial System. However, there are the following differences. Polynomials over GF(p) are represented by lists in which there are no variables, the only exponents appearing are the degrees, and zero coefficients are included.

The zero polynomial over GF(p), in any number of variables, is represented by the null list. Let $A(x)=\sum_{i=0}^{n} a_i x^i$ be any non-zero polynomial over GF(p) in r variables, with $\deg(A)=n\geq 0$ and $r\geq 1$. Thus the a_i are polynomials over GF(p) in r-1 variables or, if $r=1$, elements of GF(p). Then the list representing the polynomial A is $(n,\alpha_n,\alpha_{n-1},\ldots,\alpha_0)$ where α_i is the list representing a_i if $r>1$, and α_i is the FORTRAN integer a_i if $r=1$. Thus a non-zero polynomial in r variables is represented by a list of order r.

3. Program Descriptions

The subprograms of the SAC-1 Modular Arithmetic System are arranged below into eight categories designated by subsection headings. For each subprogram a users functional description is given, followed by an algorithm description, followed by a specification of the maximum computing time. The maximum computing time is given in O-notation as a function of appropriate parameters describing the inputs and, in some cases, the outputs of the program. For definitions and background material on such computing time analyses, see [7], [9] and [10].

3.1 Prime Number Generation

L=GENPR(A,k,m)

A is a one-dimensional integer array of length k, m is an odd FORTRAN integer, $m \geq 3$. L is the list $(p_1, p_2, \ldots, p_r)$ consisting of all prime numbers in the closed interval $[m, m+2k-2]$, with $p_1 < p_2 < \ldots < p_r$. Each p_i is represented as a FORTRAN integer. The initial values of the A(i) are ignored, and may be undefined. After execution of GENPR, A(i) =1 or 0 according as $m+2i-2$ is or is not a prime number, for $1 \leq i \leq k$.

Algorithm

(1) $n \leftarrow m+2k-2$; for $i=1,2,\ldots,k$ do: $A(i) \leftarrow 1$; $d \leftarrow 3$.

(2) If $d > \lfloor n/d \rfloor$, go to (6).

(3) (Compute the least positive integer I such that $d \mid (m+2I-2)$ and $m+2I-2 \geq 3d$.) $r \leftarrow \mathrm{rem}(m,d)$; $I \leftarrow 1$; if $r>0$ and r even, $I \leftarrow I+d-r/2$; if $r > 0$ and r odd, $I \leftarrow I+(d-r)/2$; if $m \leq d$, $I \leftarrow I+d$.

(4) For $i=I, I+d, I+2d, \ldots$ until $i > k$ do: $A(i) \leftarrow 0$.

(5) If $d \equiv 1 \pmod 6$, $d \leftarrow d+4$; if $d \not\equiv 1 \pmod 6$, $d \leftarrow d+2$; go to (2).

(6) $L \leftarrow ()$; for $i=k, k-1, \ldots, 1$ do: if $A(i)=1$, $L \leftarrow \mathrm{PFA}(m+2i-2, L)$.

Computing Time

$O(\sqrt{n}+k \log n)$, where $n=m+2k-2$.

3.2. Arithmetic Operations in GF(p)

c=CSUM(p,a,b)

p is a prime, a and b are elements of GF(p). c=a+b, the sum in GF(p).

Algorithm

(1) $c \leftarrow a+b$; if $c \geq p$, $c \leftarrow c-p$; return.

Computing Time: O(1).

c=CDIF(p,a,b)

p is a prime, a and b are elements of GF(p). c=a-b, the difference in GF(p).

Algorithm

(1) $c \leftarrow a-b$; if $c<0$, $c \leftarrow c+p$; return.

Computing Time: O(1).

c=CPROD(p,a,b)

p is a prime, a and b are elements of GF(p). $c=a \cdot b$, the product in GF(p).

Algorithm

(1) $c \leftarrow a \cdot b$ (using MPY); $c \leftarrow rem(c,p)$(using QR); return.

Computing Time: O(1).

b=CRECIP(p,a)

p is a prime, a is a non-zero element of GF(p). $b=a^{-1}$, the multiplicative inverse in GF(p).

Algorithm (The forward extended Euclidean algorithm, see [8]).

(1) $a_1 \leftarrow p$; $a_2 \leftarrow a$; $y_1 \leftarrow 0$; $y_2 \leftarrow 1$; go to (3).

(2) $q \leftarrow [a_1/a_2]$; $a_3 \leftarrow a_1 - a_2 q$; $y_3 \leftarrow y_1 - y_2 q$; $a_1 \leftarrow a_2$; $a_2 \leftarrow a_3$; $y_1 \leftarrow y_2$; $y_2 \leftarrow y_3$

(3) If $a_2 \neq 1$, go to (2); if $y_2<0$, $y_2 \leftarrow y_2+p$; $b \leftarrow y_2$; return.

Computing Time: O(log p).

b=CPOWER(p,a,n)

p is a prime number, a is an element of GF(p) and n is a non-negative FORTRAN integer. $b=a^n$, an element of GF(p).

Algorithm (Binary expansion of exponent)

(1) b←1; if n=0, return; c←a.

(2) m←[n/2]; if 2m=n, go to (3); b←CPROD(p,b,c); if m=0, return.

(3) n←m; c←CPROD(p,c,c); go to (2).

Computing Time: O(log n).

3.3 Arithmetic on Multivariate Polynomials

C=CPSUM(p,A,B)

p is a prime number, A and B are polynomials over GF(p) in r variables, r≥1. C = A+B, a polynomial over GF(p) in r variables.

Algorithm

(1) If A=0, C←BORROW(B) and return; if B=0, C←BORROW(A) and return.

(2) C←0; ADV(M,A); ADV(N,B); if M<N, interchange A and B and M and N; L←M-N; if TYPE(A)=1, go to (6).

(3) If L>0, do for i=1,2,...,L:(ADV(E,A); C←PFA(E,C)).

(4) ADV(E,A); ADV(F,B); C←PFA(CSUM(p,E,F),C).

(5) If A≠0, go to (4); go to (9).

(6) If L>0, do for i=1,2,...,L:(ADV(E,A); C←PFL (BORROW(E),C).

(7) ADV(E,A); ADV(F,B); C←PFL(CPSUM(p,E,F),C).

(8) If A≠0, go to (7).

(9) C←INV(C).

(10) If C=( ) return; if FIRST(C)≠0, go to (11); M←M-1; DECAP(E,C); go to (10).

(11) C←PFA(M,C); return.

Computing Time: $O(n_1 n_2 \cdots n_r)$, where A and B are of degree n_i or less in the i-th variable.

C=CPDIF(p,A,B)

p is a prime number, A and B are polynomials over GF(p) in r variables, $r \geq 1$. C=A−B, a polynomial over GF(p) in r variables.

Algorithm

(1) If A=0, C←CPNEG(p,B) and return; if B=0, C←BORROW(A) and return.

(2) C←0; ADV(M,A); ADV(N,B); L←M−N; T←TYPE(A); if L=0, go to (5); if L<0, go to (4).

(3) Do for i=1,2,...,L;(ADV(E,A); if T=0, C←PFA(E,C); if T=1, C←PFL(BORROW(E),C)); go to (5).

(4) L← −L; do for I=1,2,...,L;(ADV(F,B); if T=0, C←PFA(CDIF(p,0,F), C); if T=1, C←PFL(CPNEG(p,F),C)).

(5) ADV(E,A); ADV(F,B); if T=0, C←PFA(CDIF(p,E,F),C); if T=1, C←PFL(CPDIF(p,E,F),C); if A≠ (), go to (5); C←INV(C).

(6) If C=(), return; if FIRST(C) ≠ 0, go to (7); M←M−1; DECAP(E,C); go to (6).

(7) C←PFA(M,C); return.

Computing Time: $O(n_1 n_2 \cdots n_r)$, where A and B are of degree n_i or less in the i-th variable.

B=CPNEG(p,A)

p is a prime number, A is a polynomial over GF(p) in r variables, $r \geq 1$. B=−A, a polynomial over GF(p) in r variables.

Algorithm

(1) B←0; if A=0, return; ADV(N,A); B←PFA(N,B).

(2) ADV(C,A); if TYPE(A)=0, B←PFA(CDIF(p,0,C),B); if TYPE(A)=1, B←PFL(CPNEG(p,C),B); if A≠(), go to (2).

(3) B←INV(B); return.

Computing Time: $O(n_1 n_2 \cdots n_r)$, where n_i is the degree of A in the i-th variable.

C=CSPROD(p,A,b,N)

p is a prime number, A is a polynomial over GF(p) in r variables, $r \geq 1$, b is an element of GF(p), and N is a non-negative FORTRAN integer. C is the polynomial $C(X_1,\ldots,X_r)=A(X_1\ldots,X_r)\cdot b\cdot X_r^N$ where X_r is the main variable of A.

Algorithm

(1) C←(); if A=0 or b=0, return.

(2) ADV(M,A); C←PFA(M+N,0); T←TYPE(A).

(3) ADV(E,A); if T=0, C←PFA(CPROD(p,E,b),C); if T=1, C←PFL (CSPROD(p,E,b,0)); if A≠ (), go to (3); if N=0, go to (5).

(4) Do for i=1,2,...,N: (if T=0, C←PFA(0,C); if T=1, C←PFL(0,C)).

(5) C←INV(C): return.

Computing Time: $O(n_1 n_2 \cdots n_r + N)$, where n_i is the degree of A in the i-th variable.

C=CPPROD(p,A,B)

p is a prime number, A and B are polynomials over GF(p) in r variables, $r \geq 1$. C=A·B, a polynomial over GF(p) in r variables.

Algorithm

(1) C←0; if A=0 or B=0 return.

(2) ADV(N,B); if TYPE(B) ≠ 0, go to (3); F←FIRST(B); C←CSPROD(p,A,F,N); go to (5).

(3) S←A; A←TAIL(A); F←FIRST(B).

(4) ADV(E,A); G←CPPROD(p,E,F); C←PFL(G,C); if A≠0, go to (4); A←S; if N≠0, do for i=1,...,N: C←PFL(0,C); C←PFA(FIRST(A)+N,INV(C)).

(5) S←TAIL(A); T←TAIL(C).

(6) B←TAIL(B); if B=0, return; A←S; T←TAIL(T); U←T.

(7) ADV(E,A); F←FIRST(B); if TYPE(U)≠0, go to (8); H←CSUM(p,FIRST(U), CPROD(p,E,F)); go to (9).

(8) G←CPPROD(p,E,F); J←FIRST(U); H←CPSUM(p,J,G); erase J and G.

(9) ALTER(H,U); if A=0, go to (6); U←TAIL(U); go to (7).

Computing Time: $O(m_1 \cdots m_r n_1 \cdots n_r)$, where m_i is the degree of A in the i-th variable and n_i is the degree of B in the i-th variable.

3.4 Operations on Univariate Polynomials

B=CMONIC(p,A)

p is a prime number, A is a non-zero univariate polynomial over GF(p). B is the monic associate of A, a univariate polynomial over GF(p).

Algorithm

(1) a←CRECIP(p,FIRST(TAIL(A))).

(2) B←CSPROD(p,A,a,0).

Computing Time: O(n), where n=deg(A).

R=CPREM(p,A,B)

p is a prime number. A and B are non-zero univariate polynomials over GF(p). R is the remainder of A with respect to B. I.e., R is the unique polynomial over GF(p) such that, for some polynomial Q, A=B·Q+R with R=0 or deg (R)<deg(B).

Algorithm

(1) R←0; n←FIRST(B); if n=0, return; m←FIRST(A); if m<n, R←BORROW(A) and return.

(2) R← copy of A; B←TAIL(B); e←CRECIP(p,FIRST(B)); B←TAIL(B).

(3) $R(x) \leftarrow R(x)-(\mathrm{ldcf}(R)\cdot e)x^{m-n}B(x)$ (by altering the list for R).

(4) If R=0, return; m←FIRST(R); if m≥n, go to (3); return.

Computing Time: O(n(m-n+1)) if m≥n and O(1) if m<n, where m=deg(A) and n=deg(B).

C=CPGCD1(p,A,B)

p is a prime number. A and B are non-zero univariate polynomials over GF(p). C is the monic greatest common divisor of A and B, a univariate polynomial over GF(p).

Algorithm

(1) A←BORROW(A); B←BORROW(B).

(2) C←CPREM(p,A,B); ERLA(A); A←B; B←C; if B≠0, go to (2).

(3) C←CMONIC(p,A); ERLA(A); return.

Computing Time: O(n(m-k+1)), where m=max {deg(A), deg(B)}, n=min {deg(A), deg(B)} and k=deg(gcd(A,B)).

B=CPDRV(p,A)

p is a prime number. A is a univariate polynomial over GF(p). B is the derivative of A, a univariate polynomial over GF(p).

Algorithm

(1) B←0; if A=0, return; n←FIRST(A); A←TAIL(A).

(2) If n=0, go to (5); c←CPROD(p,n,FIRST(A)); n←n-1; if B≠0, go to (3); if c=0, go to (4); B←PFA(n,0).

(3) B←PFA(c,B)

(4) A←TAIL(A); go to (2).

(5) B←INV(B); return.

Computing Time: O(n), where n=deg(A).

L=CPQREM(p,A,B)

p is a prime number, A and B are non-zero univariate polynomials over GF(p) with deg(A)≥deg(B). L is the list of (Q,R) where Q and R are the unique polynomials over GF(p) such that A=B·Q+R and either R=0 or deg(R)<deg(A).

Algorithm

(1) R←0; N←FIRST(B); B←TAIL(B); if N≠0, go to (2); Q←CSPROD(p,A, CRECIP(p,FIRST(B)),0); go to (7).

(2) M←FIRST(A); Q←PFA(M-N,0); R=CSPROD(p,A,1,0); DECAP(M,R); E←CRECIP(p,FIRST(B)); B←TAIL(B).

(3) DECAP(C,R); C←CPROD(p,C,E); Q←PFA(C,Q); if C=0, go to (4); I←R; J←B; do for k=1,2,...,N: (D←CDIF(p,FIRST(I),CPROD(p,C, FIRST(J)))).

(4) M←M-1; if M≥N, go to (3).

(5) If FIRST(R)≠0, go to (6); DECAP(C,R); M←M-1; if M≥0, go to (5).

(6) If R≠0, R←PFA(M,R); Q←INV(Q).

(7) L←(Q,R); return.

<u>Computing Time:</u> $O(n(m-n+1))$, where $m=\deg(A)$ and $n=\deg(B)$.

L=CPEGCD(p,A,B)

p is a prime number, A and B are non-zero univariate polynomials over GF(p) with $\deg(A)\geq\deg(B)>0$. Let w be the resultant of A and B. If w=0, then L=0. Otherwise, L is the list (X,Y,w), where X and Y are the unique polynomials over GF(p) such that AX+BY=w, deg(X)<deg(B) and deg(Y)<deg(A).

<u>Algorithm</u>

This algorithm is based on the following theory. Let $A_1,A_2,\ldots,A_r$ be the complete polynomial remainder sequence over GF(p) defined by $A_1=A$, $A_2=B$ and $A_i=Q_iA_{i+1}+A_{i+2}$ for $1\leq i\leq r-2$. Let $c_i=\mathrm{ldcf}(A_i)$ and $n_i=\deg(A_i)$. Let $c=(-1)^s\left(\Pi_{i=1}^{r-2}c_{i+1}^{n_i-n_{i+2}}\right)c_r^{n_{r-1}-1}$, where $S=\sum_{i=1}^{r-2}n_in_{i+1}$. Then $w=cA_r$ is the resultant of A and B. Let $X_1=1$, $X_2=0$, $Y_1=0$, $Y_2=1$ and, for $1\leq i\leq r-2$, $X_{i+2}=X_i-Q_iX_{i+1}$ and $Y_{i+2}=Y_i-Q_iY_{i+1}$. Let $X=cX_r$ and $Y=cY_r$. Then X and Y are the unique polynomials over GF(p) such that AX+BY=w, deg(X)<deg(B) and deg(Y)<deg(A). (These are special cases of theorems in a forthcoming paper by G. E. Collins.)

(1) A1←BORROW(A); A2←BORROW(B); X1←PFA(0,PFA(1,0); Y2←BORROW(X1); X2←0; Y1←0; C←1; S←0.

(2) L←CPQREM(p,A1,A2); DECAP(Q,L); DECAP(A3,L); if A3=0, erase Q and go to (10).

(3) N←CPROD(2,FIRST(A1),FIRST(A2)); S←CSUM(2,S,N); D←FIRST(TAIL(A2)); N←FIRST(A1)-FIRST(A3); do for i=1,...,N: C←CPROD(p,C,D).

(4) T←CPPROD(p,X_2,Q); X3←CPDIF(p,X1,X2); erase T,X1; X1←X2; X2←X3.

(5) T←CPPROD(p,Y2,Q); Y3←CPDIF(p,Y1,Y2); erase T,X1,Q; Y1←Y2;Y2←Y3.

(6) Erase A1; A1←A2; A2←A3; if FIRST (A2)>0, go to (2).

(7) N←FIRST(A1)-1; if N=0, go to (8); D←FIRST(TAIL(A1)); do for i=1,2,...,N: C←CPROD(p,C,D).

(8) If S≠0, C←CDIF(p,0,C).

(9) X←CSPROD(p,X2,C,0); Y←CSPROD(p,Y2,C,0); w←CPROD(p,FIRST(TAIL (A2)),C); L←PFL(X,PFL(Y,PFA(w,0))).

(10) Erase A1,A2,X1,X2,Y1,Y2; return.

Computing Time: O(n(m-k+1)), where m=deg(A), n=deg(B) and k=deg (gcd(A,B)).

3.5 Erasure and Input-Output

CPERAS(A)

A is a multivariate polynomial over GF(p) for some prime number p. A is erased, i.e., the effect is the same as if ERASE were applied to A, but it is achieved more rapidly.

Algorithm

(1) If A=0, return; N←COUNT(A)-1; SCOUNT(N,A); if N>0, return; DECAP(N,A); if TYPE(A)≠0, go to (2); ERLA(A); return.

(2) N←COUNT(A)-1; SCOUNT(N,A); if N>0, return; DECAP(C,A); CPERAS(C); if A≠(), go to (2); return.

Computing Time: $O(n_1 n_2 \cdots n_r)$, where n_i is the degree of A in the i-th variable.

In the following input and output programs, the same external canonical form is used as for polynomials over the integers I. In fact the subprograms PREAD and PWRITE of the SAC-1 Polynomial System are used for the actual input and output, and CPMOD and CPGARN (see Section 3.6) are used for conversion between the two internal canonical forms.

A=CPREAD(p,U)

p is a prime number, U is a logical unit number. The polynomial A, a multivariate polynomial over GF(p) is read from unit U. If U was initially positioned at an end-of-file, then A=-1. If U was initially positioned at a polynomial which is syntactically incorrect, then A=-2. For further details, see the SAC-1 Polynomial System manual [5].

Algorithm

(1) A←PREAD(U); if A<0, return.

(2) B←A; A←CPMOD(p,B); PERASE(B); return.

CPWRIT(p,U,A,L)

p is a prime number, U is a logical unit number, A is a multivariate polynomial over GF(p), L is a list of variables $(X_1,...,X_r)$ to be used in the external canonical form, with X_r as the main variable of A. A is converted to external canonical form and written on unit U. For further details, see the SAC-1 Polynomial System manual [5].

Algorithm

(1) Q←PFA(1,0); B←CPGARN(Q,0,2p+1,A,L); PWRITE(U,B).

(2) ERLA(Q); PERASE(B); return

3.6 Reduction Modulo p and Chinese Remainder Theorem

There is a unique homomorphism from the ring I of integers onto GF(p), which we denote by $\emptyset_p$. If A is a polynomial over I in r variables, $A(X_1,...,X_r)=\sum_{i=0}^{n} A_i(X_1,...,X_{r-1})\cdot X_r^i$, we define by induction on r, $\emptyset_p(A(X_1,...,X_r))=\sum_{i=0}^{n} \emptyset_p(A_i(X_1,...,X_{r-1})\cdot X_r^i$.

a=CMOD(p,I)

p is a prime number, I is an infinite precision integer. a= $\emptyset_p$(I), an element of GF(p).

Algorithm

(1) T←PFA(p,0); R←IREM(I,T); ERLA(T); a←0; if R=0, return; a←FIRST(R); if a<0, a←a+p; ERLA(R); return.

Computing Time: O(log|I|)

B=CPMOD(p,A)

p is a prime number, A is a multivariate polynomial over the ring of integers, or an infinite-precision integer. B=$\emptyset_p$ (A), a multivariate polynomial over GF(p), or an element of GF(p).

Algorithm

(1) B←0; if A=0, return; if A is an integer, B←CMOD(p,A) and return.

(2) A←TAIL(A); N←FIRST(TAIL(A)); T←TYPE(FIRST(A)); M←N.

(3) If A=() or M≠ FIRST(TAIL(A)), C←0 and go to (5).

(4) E←FIRST(A); A←TAIL(TAIL(A)); C←CPMOD(p,E).

(5) If T=0, B←PFA(C,B); if T≠0, B←PFL(C,B); M←M-1; if M≥0, go to (3); B←INV(B).

(6) If FIRST(B)≠0, go to (7); DECAP(C,B); N←N-1; if N≥0, go to (6).

(7) If B≠0, B←PFA(N,B); return.

Computing Time: $O(n_1 n_2 \cdots n_r \log N)$, where n_i is the degree of A in the i-th variable and N is an upper bound on all the numerical coefficients of A, or O(log|A|) if A is an integer.

C=CGARN(Q,B,p,a)

p is an odd prime number, a is an element of GF(p). Q is an odd positive infinite-precision integer which is relatively prime to p. B is an infinite-precision integer such that $|B|<Q/2$. C is the unique infinite-precision integer such that $C\equiv a \pmod{p}$, $C\equiv B \pmod{Q}$ and $|C|<pQ/2$. Note that if Q=1 and B=0 one obtains the integer C such that $C\equiv a \pmod{p}$ and $|C|<p/2$.

<u>Algorithm</u> (This is Garner's algorithm for the case of two congruences; see [10], pp. 253-256.)

(1) b←CMOD(p,B); q←CMOD(p,Q).

(2) d←(a-b)/q (mod p arithmetic); if d>p/2, d←d-p.

(3) C←dQ+B; return.

<u>Computing Time</u>: O(log Q).

C=CPGARN(Q,B,p,A,L)

P is an odd prime number, A is a polynomial in r variables over GF(p), $r\geq 0$. Q is an odd positive infinite-precision integer relatively prime to p. B is a polynomial over the integers in r variables, whose coefficients are less than Q/2 in magnitude. L is a list $(X_1,\ldots X_r)$ of r variables. (If r=0, then B is an integer, A is an element of GF(p) and L is the null list.) C is the unique polynomial $C(X_1,\ldots,X_r)$, X_r the main variable, over the integers such that $C\equiv B \pmod{Q}$, $C\equiv A \pmod{p}$, and the coefficients of C are less than pQ/2 in magnitude.

Note that if Q=1 and B=0, one obtains the polynomial C such that $C\equiv A \pmod{p}$ and the coefficients of C are less than p/2 in magnitude.

<u>Algorithm</u>

(1) If L=(), C←CGARN(Q,B,p,A,L) and return.

(2) L←CINV(L); C←CPGARN(Q,B,p,A,L) using recursive procedure; erase L; return.

(3) (Begin recursive procedure C=CPGARN(Q,B,p,A,L)) C←0; M←-1; if A≠ (), ADV(M,A); N←-1; if B≠ (), B←TAIL(B), N←FIRST(TAIL(B)).

(4) If A=0 and B=0, go to (10); E←0; F←0.

(5) If M≥N, go to (6); K←M; M←M-1; ADV(E,A).

(6) If K>N, go to (7); K←N; ADV(F,B); B←TAIL(B); N←-1; if B≠0, N←FIRST(TAIL(B)).

(7) If E=0 and F=0, go to (4).

(8) If TAIL(L)=(), G←CGARN(Q,F,p,E); otherwise, G←CPGARN(Q,F,p,E, TAIL(L)) using recursive procedure.

(9) C←PFA(K,PFL(G,C)); go to (4).

(10) If C≠(), C←PFL(BORROW(FIRST(L)),INV(C)); recursive procedure return.

Computing Time: $O(n_1 n_2 \cdots n_r \log Q)$, where A and B are polynomials of degree n_i or less in the i-th variable.

3.7 Evaluation and Interpolation

C=CPEVAL(p,A,b)

p is a prime number, A is a polynomial in r variables over GF(p), $r \geq 1$, and b is an element of GF(p). C is the polynomial $C(X_1,\ldots,X_{r-1}) = A(X_1,\ldots,X_{r-1},b)$, a polynomial over GF(p) in r-1 variables (or, if r=1, an element of GF(p)), where x_r is the main variable of A.

Algorithm

(1) C←0; if A=0, return; A←TAIL(A); if TYPE(A)=1, go to (3); ADV(C,A).

(2) If A=(), return; ADV(E,A); C←CSUM(p,CPROD(p,C,b),E); go to (2).

(3) d←1; A←CINV(A); DECAP(C,A).

(4) If A=(), return; d←CPROD(p,d,b); DECAP(E,A); if E=0, go to (4); F←CSPROD(p,E,d,0); G←CPSUM(p,C,F); erase C, E and F; C←G; go to (4).

Computing Time: $O(n_1 n_2 \cdots n_r)$, where n_i is the degree of A in the i-th variable.

G=CPINT(p,A,b,C,D,r)

p is a prime number, b is an element of GF(p), r is a positive FORTRAN integer, and A is a polynomial over GF(p) in r variables, $r \geq 1$. D is a univariate polynomial over GF(p) which is a product of k+1 distinct monic linear factors, $D(X)=\Pi_{i=0}^{k}(X-b_i)$, and $b \neq b_i$ for $0 \leq i \leq k$. C is a polynomial over GF(p) in r-1 variables (or, if r=1, an element of GF(p)). The degree of A in its main variable, X_r, is at most k. G is the unique interpolating polynomial in r variables over GF(p), of degree k+1 or less in X_r, such that $G(X_1,\ldots,X_{r-1},b_i)=A(X_1,\ldots,X_{r-1},b_i)$ for $0 \leq i \leq k$ and $G(X_1,\ldots,X_{r-1},b)=C(X_1,\ldots,X_{r-1})$.

Note that a special case k= -1 is permitted in CPINT, in which D(X)=1 (a constant polynomial) and A=0. Then $G(X_1,\ldots,X_r)=C(X_1,\ldots,X_{r-1})$, i.e., G is a polynomial in r variables which is of degree 0 in X_r.

Algorithm (The formula $G(X_1,\ldots,X_r)=\{C(X_1,\ldots,X_{r-1})-A(X_1,\ldots,X_{r-1},b)\} D(b)^{-1} D(X_r)+A(X_1,\ldots,X_r)$ is used.)

(1) T←CPEVAL(p,A,b); v←CRECIP(p,CPEVAL(p,D,b)); if r>1, go to (3).

(2) u←CDIF(p,C,T); w←CPROD(p,u,v); V←CSPROD(p,D,w,0); go to (4).

(3) U←CDIF(p,C,T); erase T; W←CSPROD(p,U,v,0); erase U; V←W·D (by repeated applications of CSPROD; erase W.

(4) G←CPSUM(p,V,A); erase V; return.

Computing Time: $O(n_1 n_2 \cdots n_{r-1} k)$, where A and C are polynomials of degree n_i or less in the i-th variable, and k is defined above.

3.8 Univariate Polynomial Factorization

The following nine subprograms all pertain to the factorization of a univariate polynomial over a finite field GF(p). Clearly it suffices to factor monic polynomials and a method is described by Knuth in [10], page 381, for expressing any polynomial over GF(p) as a product of squarefree polynomials. This method is not realized by a subprogram in the present system, although it could be easily added.

The subprogram CPBERL computes the complete factorization of any monic squarefree polynomial A over GF(p), i.e., it produces a list of all the monic irreducible factors of A. CPBERL implements the Berlekamp factorization algorithm ([10], pages 381-389). Since the computing time for Berlekamp's algorithm is roughly $O(pn^3)$, this subprogram will be practicable only for moderately small values of p.

For larger values of p the subprogram CPDDF may be useful, since its maximum computing time is only $O(n^3+n^2(\log p))$. However, CPDDF does not always produce a complete factorization-it produces a "distinct degree factorization." I.e., if A is the monic squarefree polynomial to be factored, it produces a list $((n_1,A_1),\ldots,(n_k,A_k))$ where the n_i are positive integers, $n_1<n_2<\cdots<n_k$, and A_i is the product of all monic irreducible factors of A which are of degree n_i. Thus $A=A_1\cdots A_k$ and this is a complete factorization just in case no two irreducible factors of A have the same degree.

The other seven subprograms are present primarily for use by CPBERL and CPDDF. However three of them, CVPROD, CMPROD and CMNULL, perform standard linear algebra operations over a field GF(p), and may therefore have a general usefulness.

These nine subprograms represent just a first step towards the development of a SAC-1 system for the factorization of polynomials over the integers. But they are included in the present system also for their possible usefulness in algebraic coding theory and other disciplines where the factorization of polynomials over a finite field has rather immediate applicability.

F=CPBERL(p,A)

p is a prime number and A is a monic squarefree univariate polynomial over GF(p), with $\deg(A)\geq 2$. F is a list $(A_1,\ldots,A_r)$ of all the monic irreducible factors of A of positive degree.

Algorithm (See [10], pp. 381-389.)

(1) Q←CPBQ(p,A); n←deg(A).

(2) CPTOM(n,Q).

(3) Q←Q-I, where I is the n by n identity matrix.

(4) CMNULL(p,Q).

(5) Convert the matrix Q into a list $B=(B_1,\ldots,B_r)$ of monic polynomials over GF(p), where $B_i(X)=\sum_{j=o}^{n-1} Q_{r-i+1,j+1}X^j$ so that $0=\deg(B_1)<\deg(B_2)<\ldots<\deg(B_r)$.

(6) F←CPBG(p,A,B,1); erase B; return.

Computing Time: $O(n^3+rn^2p+rnp(\log p))$, where $n=\deg(A)$ and r is the number of monic irreducible factors of A of positive degree.

Note that in the present section, computing times are based on the assumption that, since p is single-precision, the time for computing a multiplicative inverse in GF(p) is $O(\log p)$, while the computing time for any other arithmetic operation in GF(p) is $O(1)$.

L=CPDDF(p,A)

p is a prime number and A is a monic squarefree polynomial over GF(p), $\deg(A)\geq 2$. L is a list $((n_1,A_1),\ldots,(n_k,A_k))$ where each n_i is a positive integer, $n_1<n_2<\cdots<n_k$, and A_i is the product of all irreducible monic factors of A of degree n_i, a polynomial of positive degree, for $1\leq i\leq k$.

<u>Algorithm</u> (see [10], pp. 389-390)

(1) Q←CPBQ(p,A); k←1; B←BORROW(FIRST(TAIL(Q))); C←BORROW(A); L←().

(2) W(X)←B(X)-X; D←gcd(W,C); erase W; if deg(D)=0,go to (3); P←(k,D); L←PFL(P,L); C←C/D.

(3) Erase D; k←k+1; if deg(C)≥2k, go to (5).

(4) If deg(C)>0, P←(deg(C),C) and L←PFL(P,L); if deg(C)=0, erase C; erase B and Q; return.

(5) Convert B to a vector V; if k>2, go to (6); n←deg(A); CPTOM(n,Q).

(6) Multiply the vector V by the matrix Q; B←polynomial represented by the vector V; go to (2).

<u>Computing Time</u>: $O(n^3+n^2(\log p))$, where n=deg(A).

S=CPBG(p,A,B,d)

p is a prime number, A is a monic squarefree univariate polynomial over GF(p). B is a list $(B_1,\ldots,B_r)$ of monic univariate polynomials over GF(p), which constitute a basis for the space of all polynomials C of degree less than deg(A) such that A is a divisor of C^p-C. It is further required that $B_1=1$ and $B_i(0)=0$ for $2\leq i\leq r$. d is a positive FORTRAN integer such that A has no irreducible factors of degree less than d. S is a list $(A_1,\ldots,A_r)$ consisting of all the monic irreducible factors of A of positive degree, with $\deg(A_1)\geq\deg(A_2)\geq\cdots\geq\deg(A_r)$.

<u>Algorithm</u>

(1) S←PFL(BORROW(A),0); r←LENGTH(B); if r=1, return.

(2) B←TAIL(B); m←1.

(3) T←(); BI←FIRST(B); B←TAIL(B).

(4) DECAP(C,S); if deg(C)=d, go to (6).

(5) For j=0,1,...,p-1,do:
   a. G←gcd(BI-j,C).
   b. If deg(G)=0, go to (f).
   c. If deg(G)=deg(C), go to (6).

d. CPINS(G,T); C←C/G; m←m+1; if m=r, go to (7).

e. If deg(C)=d, go to (6).

f. Continue.

(6) CPINS(C,T); if S≠(), go to (4); S←T; go to (3).

(7) Insert C and each remaining polynomial in S into T, using CPINS; S←T; return.

Computing Time: $O(rn^2p+rnp(\log p))$, where n=deg(A) and r is the number of monic irreducible factors of A of positive degree or, equivalently, the number of polynomials in the basis B.

Q=CPBQ(p,A)

p is a prime number and A is a univariate polynomial over GF(p) with deg(A)≥2. Q is the list $(Q_o,Q_1,\ldots,Q_{n-1})$, where Q_i is the univariate polynomial $Q_i(x)=\mathrm{rem}(x^{pi},A(x))$ and n=deg(A).

Algorithm

(1) $L\leftarrow 2^k$, where $k=[\log_2 p]$.

(2) Compute $B(x)=Q_1(x)$ as follows:

(a) B(x)←x; M←p−L; L←[L/2].

(b) $B\leftarrow \mathrm{rem}(B^2,A)$; if M<L, go to (c); M←M−L; B(x)←rem(x·B(x),A(x)).

(c) L←[L/2]; if L≠0, go to (b).

(3) C(x)←1; Q←(C); N←deg(A)−1; for i=1,...,N do: (C←rem(B·C,A); Q←PFL(C,Q)); Q←INV(Q).

Computing Time: $O(n^3+n^2(\log p)+(\log p)^2)$, where n=deg(A). (Note that the term $(\log p)^2$ may be omitted if A is monic.)

CPTOM(n,L)

n is a positive FORTRAN integer. L is a list $(Q_o,Q_1,\ldots,Q_{n-1})$ of non-zero univariate polynomials over GF(p) of degrees less than n. If $Q_i(x)=\sum_{j=o}^{n-1} q_{ij}\, x^j$, then after execution of CPTOM the new value of

L is the list $((q_{o,o},\ldots,q_{n-1,o}),\ldots,(q_{o,n-1},\ldots, q_{n-1,n-1}))$. No overlapping is permitted in the representation of the input list L, since it is altered to produce the representation of the output list.

Algorithm:

(1) Convert each polynomial $Q_i=(k,q_{ik},\ldots,q_{io})$ on L to a list of length n by prefixing n-1-k zeros to $(q_{ik},\ldots,q_{io})$.

(2) Convert the resulting list $L = ((q_{o,n-1},\ldots,q_{o,o}),\ldots,(q_{n-1,n-1},\ldots,q_{n-1,o}))$ to the form $((q_{o,o},\ldots,q_{n-1,o}),\ldots,(q_{o,n-1},\ldots,q_{n-1,n-1}))$:

(a) IL←INV(L); L←().

(b) R←IL; C←().

(c) Q←FIRST(R); if Q=0, then erase IL and return.

(d) ALTER(TAIL(Q),R); SSUCC(C,Q): C←Q; R←TAIL(R); if R≠0, go to (c).

(e) Prefix C to L and go to (b).

Computing Time: $O(n^2)$.

CMNULL(p,L)

p is a prime number and L is a second-order list representing an m by n matrix M over GF(p) by columns. I.e., $L=(L_1,\ldots,L_n)$ and $L_j=(M_{1j},M_{2j},\ldots,M_{mj})$. After execution of CMNULL, L is a list $(v_1,v_2,\ldots,v_r)$ representing a basis for the null space of M, consisting of all x such that xM=0. Each v_i is a list $(v_{i1},v_{i2},\ldots,v_{im})$ representing a 1 by m row vector. r is the dimension of the null space, $0\le r\le m$. If r>0 and k_i is the largest integer k such that $v_{ik}\neq o$, then $k_1>k_2>\ldots>k_r$ and $v_{ik_i}=1$ for $1\le i\le r$. (Equivalently, if V is the r by m matrix $V_{ij}=v_{i,m-j+1}$, then V is in row-echelon form.) No overlapping is permitted in the representation of the input list L, since it is altered and erased.

Alternatively, if the input list L represents a matrix M by rows, then the output of CMNULL is a basis for the null space consisting of all x such that Mx=0.

Algorithm

Elementary column operations are performed on the matrix represented by L to reduce it to a triangular form (similar to standard column-echelon form), from which basis vectors for its null space can easily be found.

(1) A←L, B←(), L←(), m←LENGTH(FIRST(A)).

(2) Perform steps (3)-(6) for k=1,...,m.

(3) Search list A for a pivot column:

  (a) S←A, A←().
  (b) If S is empty, then go to (5).
  (c) Remove the first column T from S, and remove the first element d from T. If d≠0, then go to (4).
  (d) Prefix column T to A and go to (3b).

(4) Prefix column T to B and perform the following pivot operations:

  (a) Compute e← $-d^{-1}$ (in GF(p)).
  (b) Multiply each element of T by e.
  (c) If S is empty then go to (6).
  (d) Remove the next column Q from S, remove the first element d from Q, and prefix Q to A. If d≠0, add d times column T to column Q. Go to (4c).

(5) No pivot column was found in step (3), hence generate a null space basis vector v as follows:

  (a) Let v←(1,0,...,0), where there are m-k zeros. Let C←B.
  (b) If C is empty, then go to (5d).
  (c) Compute the inner product d of the vectors v and FIRST(C), and prefix d to v. Let C←TAIL(C) and go to (5b).
  (d) Prefix v to L and () to B.

(6) Continue.

(7) Erase A and B and return.

Computing Time: O(kmn+rmn+k(log p)), where L is an m by n matrix over GF(p), k=min(m,n) and r is the dimension of the null space of L.

c=CVPROD(p,A,B)

p is a prime number, A and B are first order lists over GF(p). c is the inner product of A and B considered as vectors. Specifically, if A or B is null then c=0. Otherwise, let $A=(a_1,\ldots,a_m)$, $B=(b_1,\ldots,b_n)$ and k=min(m,n). Then $c = \sum_{i=1}^{k} a_i b_i$ in GF(p).

Algorithm

(1) c←0.

(2) If A=0 or B=0, return.

(3) c←c+FIRST(A)·FIRST(B); A←TAIL(A); B←TAIL(B); go to (2).

Computing Time: O(min(m,n)).

C=CMPROD(p,A,B)

p is a prime number, A and B are second order lists over GF(p). C is the matrix product of A and B, i.e., if A represents an m by n matrix by rows and B represents an n by q matrix by columns, then C represents their product by rows. More generally, if $A=(A_1,\ldots,A_m)$ and $B=(B_1,\ldots,B_q)$, then $C=((c_{1,1},\ldots,c_{1,q}),\ldots,(c_{m,1},\ldots,c_{m,q}))$ where $c_{i,j}$=CVPROD(p,A_i,B_j).

Algorithm

(1) Q←A; C←(); BI←CINV(B).

(2) If Q=0, go to (5); R←FIRST(Q); S←BI; V←().

(3) If S=0, go to (4); V←PFA(CVPROD(p,R,FIRST(S)),V); S←TAIL(S); go to (3).

(4) C←PFL(V,C); Q←TAIL(Q); go to (2).

(5) C←INV(C); erase BI; return.

Computing Time: O(mnq), if A represents an m by n matrix and B an n by q matrix.

CPINS(A,L)

A is a univariate polynomial over GF(p) and L is the empty list or is a list $(A_1,\ldots,A_k)$ of univariate polynomials over GF(p) with deg $(A_1)\geq deg(A_2)\geq\ldots\geq deg(A_k)$. After execution of CPINS, L is the same list with A inserted following the last A_i (if any) such that $deg(A_i) > deg(A)$.

<u>Algorithm</u>

(1) N←L.

(2) If N=(), go to (3); if deg(A)≥deg(FIRST(N)), go to (3); M←N; N←TAIL(N); go to (2).

(3) B←PFL(A,N); if N=L, let L←B; otherwise, SSUCC(B,M); return.

<u>Computing Time:</u> O(k), where k=LENGTH(L).

4. REFERENCES

1. American Standards Association. A Programming Language for Information Processing on Automatic Data Processing Systems. C.A.C.M., Volume 7, No. 10 (October, 1964), pp. 591-625.

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

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

4. Collins, G. E. and Pinkert, James R. The Revised SAC-1 Integer Arithmetic System. University of Wisconsin Computing Center, Technical Reference No. 9, November, 1968.

5. Collins, G. E. The SAC-1 Polynomial System. University of Wisconsin Computing Center, Technical Reference No. 2, January, 1968.

6. Collins, G. E. The SAC-1 Rational Function System. University of Wisconsin Computing Center, Technical Reference No. 8, July, 1968.

7. Collins, G. E. Computing Time Analyses of Some Arithmetic and Algebraic Algorithms. University of Wisconsin Computer Sciences Department Technical Report No. 36, July, 1968. To be published in Proceedings of the IBM 1968 Summer Institute on Symbolic Mathematics by Computer.

8. Collins, G. E. Computing Multiplicative Inverses in GF(p). Mathematics of Computation, Volume 23, No. 105 (January, 1969), pp. 197-200.

9. Knuth, D. E. The Art of Computer Programming, Volume 1: Fundamental Algorithms, Addison-Wesley, 1968.

10. Knuth, D. E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison-Wesley, 1969.

5. FORTRAN Program Listings

```
      INTEGER FUNCTION CDIF(P,X,Y)
      INTEGER P,X,Y
      CDIF=X-Y
      IF (CDIF.LT.0) CDIF=CDIF+P
      RETURN
      END


      INTEGER FUNCTION CGARN(Q,B,P,A)
      INTEGER A,B,C,D,E,P,Q
      INTEGER BORROW,CDIF,CMOD,CPROD,CRECIP,IPROD,ISUM,PFA
      E=CMOD(P,B)
      E=CDIF(P,A,E)
      IF (E.NE.0) GO TO 1
      C=BORROW(B)
      GO TO 2
1     D=CMOD(P,Q)
      D=CRECIP(P,D)
      D=CPROD(P,D,E)
      IF (D.GT.P/2) D=D-P
      E=PFA(D,0)
      D=IPROD(Q,E)
      CALL ERLA(E)
      C=ISUM(D,B)
      CALL ERLA(D)
2     CGARN=C
      RETURN
      END


      SUBROUTINE CMNULL(P,L)
      INTEGER CPROD,CRECIP,CSUM,CVPROD,FIRST,LENGTH,PFA,PFL,TAIL
      INTEGER P,L,A,B,C,D,I,J,K,M,MDI,Q,R,S,T,U,V
1     A = L
      B = 0
      L = 0
      M = LENGTH(FIRST(A))
2     DO 6 K = 1,M
3     S = A
      A = 0
31    IF (S .EQ. 0) GO TO 5
      CALL DECAP(T,S)
      CALL DECAP(D,T)
      IF (D .NE. 0) GO TO 4
      A = PFL(T,A)
      GO TO 31
4     B = PFL(T,B)
      MDI = P-CRECIP(P,D)
      U = T
41    IF (U .EQ. 0) GO TO 42
      CALL ALTER(CPROD(P,FIRST(U),MDI),U)
      U = TAIL(U)
```

```
      GO TO 41
42    IF (S .EQ. 0) GO TO 6
      CALL DECAP(Q,S)
      CALL DECAP(D,Q)
      A = PFL(Q,A)
      IF (D .EQ. 0) GO TO 42
      U = T
      R = Q
43    IF (U .EQ. 0) GO TO 42
      CALL ALTER(CSUM(P,FIRST(R),CPROD(P,D,FIRST(U))),R)
      U = TAIL(U)
      R = TAIL(R)
      GO TO 43
5     V = 0
      J = M-K
51    IF (J .EQ. 0) GO TO 52
      DO 511 I = 1,J
511   V = PFA(0,V)
52    V = PFA(1,V)
      C = B
53    IF (C .EQ. 0) GO TO 54
      V = PFA(CVPROD(P,FIRST(C),V),V)
      C = TAIL(C)
      GO TO 53
54    L = PFL(V,L)
      B = PFL(0,B)
6     CONTINUE
7     CALL ERASE(A)
      CALL ERASE(B)
      RETURN
      END


      INTEGER FUNCTION CMOD(P,A)
      INTEGER P,A,R,T
      INTEGER PFA,IREM,FIRST
      T=PFA(P,0)
      R=IREM(A,T)
      CALL ERLA(T)
      CMOD=0
      IF (R.EQ.0) RETURN
      CMOD=FIRST(R)
      IF (CMOD.LT.0) CMOD=CMOD+P
      CALL ERLA(R)
      RETURN
      END
```

```
      INTEGER FUNCTION CMONIC(P,X)
      INTEGER P,X,A
      INTEGER FIRST,TAIL,CRECIP,CSPROD
      A=CRECIP(P,FIRST(TAIL(X)))
      CMONIC=CSPROD(P,X,A,0)
      RETURN
      END
```

```
      INTEGER FUNCTION CMPROD(P,A,B)
      INTEGER CINV,CVPROD,FIRST,INV,PFA,PFL,TAIL
      INTEGER P,A,B,BI,L,Q,R,S,V
1     Q = A
      L = 0
      BI = CINV(B)
2     IF (Q .EQ. 0) GO TO 5
      R = FIRST(Q)
      S = BI
      V = 0
3     IF (S .EQ. 0) GO TO 4
      V = PFA(CVPROD(P,R,FIRST(S)),V)
      S = TAIL(S)
      GO TO 3
4     L = PFL(V,L)
      Q = TAIL(Q)
      GO TO 2
5     CMPROD = INV(L)
      CALL ERASE(BI)
      RETURN
      END
```

```
      INTEGER FUNCTION CPBERL(P,A)
      INTEGER CDIF,CPBG,CPBQ,FIRST,INV,PFA,PFL,TAIL
      INTEGER P,A,B,BP,BV,D,I,J,K,N,Q,S,T
1     Q = CPBQ(P,A)
2     N = FIRST(A)
      CALL CPTOM(N,Q)
3     S = Q
      I = 1
32    T = FIRST(S)
      J = 1
34    IF (J .EQ. I) GO TO 36
      J = J+1
      T = TAIL(T)
      GO TO 34
36    CALL ALTER(CDIF(P,FIRST(T),1),T)
      I = I+1
      S = TAIL(S)
      IF (S .NE. 0) GO TO 32
4     CALL CMNULL(P,Q)
5     B = 0
```

```
52      CALL DECAP(BV,Q)
        BV = INV(BV)
        DO 54 K = 1,N
        CALL DECAP(D,BV)
        IF (D .NE. 0) GO TO 56
54      CONTINUE
56      BP = PFA(N-K,PFA(D,BV))
        B = PFL(BP,B)
        IF (Q .NE. 0) GO TO 52
6       CPBERL = CPBG(P,A,B,1)
        CALL ERASE(B)
        RETURN
        END
```

```
        INTEGER FUNCTION CPBG(P,A,B,D)
        INTEGER BORROW,CPGCD1,CPQREM,FIRST,LENGTH,PFL,TAIL
        INTEGER P,A,B,D,BI,BIO,BT,C,DC,DG,E,G,J,M,R,S,T,TEMP,TEMP1
1       S = PFL(BORROW(A),0)
        R = LENGTH(B)
        IF (R .EQ. 1) GO TO 8
2       BT = TAIL(B)
        M = 1
3       T = 0
        CALL ADV(BI,BT)
        E = TAIL(BI)
35      BIO = E
        E = TAIL(BIO)
        IF (E .NE. 0) GO TO 35
4       CALL DECAP(C,S)
        DC = FIRST(C)
        IF (DC .EQ. D) GO TO 6
5       J = P
51      G = CPGCD1(P,BI,C)
        DG = FIRST(G)
        IF (DG .GT. 0) GO TO 52
        CALL ERLA(G)
        GO TO 54
52      IF (DG .LT. DC) GO TO 53
        CALL ERLA(G)
        GO TO 6
53      CALL CPINS(G,T)
        TEMP = CPQREM(P,C,G)
        CALL ERLA(C)
        CALL DECAP(C,TEMP)
        CALL DECAP(TEMP1,TEMP)
        M = M+1
        IF (M .EQ. R) GO TO 7
        DC = FIRST(C)
        IF (DC .EQ. D) GO TO 6
54      J = J-1
        CALL ALTER(J,BIO)
        GO TO 51
```

```
6     CALL CPINS(C,T)
      CALL ALTER(0,BIO)
      IF (S .NE. 0) GO TO 4
      S = T
      GO TO 3
7     CALL ALTER(0,BIO)
71    CALL CPINS(C,T)
      IF (S .EQ. 0) GO TO 72
      CALL DECAP(C,S)
      GO TO 71
72    S = T
8     CPRG = S
      RETURN
      END
```

```
      INTEGER FUNCTION CPBQ(P,A)
      INTEGER PFA,PFL,INV,FIRST,CPPROD.CSPROD,CPREM
      INTEGER P,A,B,C,D,Q
      L = 2
1     L = L+L
      IF (L .LE. P) GO TO 1
      L = L/2
      B = PFA(1,PFA(1,PFA(0,0)))
      M = P-L
      L = L/2
2     C = CPPROD(P,B,B)
      CALL ERLA(B)
      B = CPREM(P,C,A)
      CALL ERLA(C)
      IF (M .LT. L) GO TO 3
      M = M-L
      C = CSPROD(P,B,1,1)
      CALL ERLA(B)
      B = CPREM(P,C,A)
      CALL ERLA(C)
3     L = L/2
      IF (L .NE. 0) GO TO 2
      C = PFA(0,PFA(1,0))
      Q = PFL(C,0)
      N = FIRST(A)-1
      DO 4 I = 1,N
      D = CPPROD(P,B,C)
      C = CPREM(P,D,A)
      CALL ERLA(D)
4     Q = PFL(C,Q)
      CALL ERLA(B)
      CPBQ = INV(Q)
      RETURN
      END
```

```
      INTEGER FUNCTION CPDDF(P,A)
      INTEGER BORROW,CINV,CPBQ,CPGCD1,CPQREM,CPSUM,CVPROD,FIRST,INV,PFA
      INTEGER PFL,TAIL,P,A,B,BV,C,D,DC,J,K,L,MX,N,Q,R,TEMP,TEMP1,W
1     Q = CPBQ(P,A)
      K = 1
      B = BORROW(FIRST(TAIL(Q)))
      C = BORROW(A)
      L = 0
      MX = PFA(1,PFA(P-1,PFA(0,0)))
2     W = CPSUM(P,B,MX)
      IF (W .NE. 0) GO TO 22
      D = BORROW(C)
      GO TO 24
22    D = CPGCD1(P,W,C)
      CALL ERLA(W)
      IF (FIRST(D) .EQ. 0) GO TO 3
24    L = PFL(PFA(K,PFL(D,0)),L)
      TEMP = CPQREM(P,C,D)
      CALL ERLA(C)
      CALL DECAP(C,TEMP)
      CALL DECAP(TEMP1,TEMP)
      GO TO 31
3     CALL ERLA(D)
31    K = K+1
      DC = FIRST(C)
4     IF (DC .GE. 2*K) GO TO 5
      IF (DC .GT. 0) GO TO 45
      CALL ERLA(C)
      GO TO 7
45    L = PFL(PFA(DC,PFL(C,0)),L)
      GO TO 7
5     IF (K .GT. 2) GO TO 6
      BV = CINV(TAIL(B))
      CALL ERLA(B)
      N = FIRST(A)
      CALL CPTOM(N,Q)
      GO TO 61
6     CALL DECAP(TEMP,B)
      BV = INV(B)
61    B = 0
      R = Q
62    B = PFA(CVPROD(P,BV,FIRST(R)),B)
      R = TAIL(R)
      IF (R .NE. 0) GO TO 62
      CALL ERLA(BV)
      DO 65 J = 1,N
      CALL DECAP(TEMP,B)
      IF (TEMP .NE. 0) GO TO 67
65    CONTINUE
67    B = PFA(N-J,PFA(TEMP,B))
      GO TO 2
```

```
7       CALL ERASE(Q)
        CALL ERLA(MX)
        CALL ERLA(B)
        CPDDF = L
        RETURN
        END


        INTEGER FUNCTION CPDIF(P,AA,BB)
        INTEGER AA,A,BB,B,C,E,F,G,I,L,M,N,P,R,T
        INTEGER BORROW,CDIF,CPNEG,FIRST,INV,PFA,PFL,TYPE
        A=AA
        B=BB
        R=1
        GO TO 2
 19     CPDIF=C
        RETURN
C     RECURSIVE PROCEDURE C=CPDIF(P,A,B).
 2      IF (A.NE.0) GO TO 3
        C=CPNEG(P,B)
        GO TO 17
 3      IF (B.NE.0) GO TO 4
        C=BORROW(A)
        GO TO 17
 4      C=0
        CALL ADV(M,A)
        CALL ADV(N,B)
        L=M-N
        T=TYPE(A)
        IF (L.EQ.0) GO TO 10
        IF (L.LT.0) GO TO 7
        DO 6 I=1,L
        CALL ADV(E,A)
        IF (T.EQ.0) GO TO 5
        C=PFL(BORROW(E),C)
        GO TO 6
 5      C=PFA(E,C)
 6      CONTINUE
        GO TO 10
 7      L=-L
        M=N
        DO 9 I=1,L
        CALL ADV(F,B)
        IF (T.EQ.0) GO TO 8
        C=PFL(CPNEG(P,F),C)
        GO TO 9
 8      C=PFA(CDIF(P,0,F),C)
 9      CONTINUE
 10     IF (T.EQ.1) GO TO 12
 11     CALL ADV(E,A)
        CALL ADV(F,B)
        C=PFA(CDIF(P,E,F),C)
        IF (A.NE.0) GO TO 11
```

```
      GO TO 14
 12   CALL ADV(E,A)
      CALL ADV(F,B)
C    RECURSIVE CALL G=CPDIF(P,E,F).
      CALL STACK3(A,B,C)
      CALL STACK2(M,R)
      A=E
      B=F
      R=2
      GO TO 2
 13   G=C
      CALL UNSTK2(M,R)
      CALL UNSTK3(A,B,C)
C    END RECURSIVE CALL.
      C=PFL(G,C)
      IF (A.NE.0) GO TO 12
 14   C=INV(C)
 15   IF (C.EQ.0) GO TO 17
      IF (FIRST(C).NE.0) GO TO 16
      M=M-1
      CALL DECAP(E,C)
      GO TO 15
 16   C=PFA(M,C)
 17   GO TO (19,13), R
C    END RECURSIVE PROCEDURE.
      END
```

```
      INTEGER FUNCTION CPDRV(P,A)
      INTEGER A,B,C,P,T
      INTEGER FIRST,INV,PFA,TAIL,CPROD
1     B=0
      IF (A.EQ.0) GO TO 5
      N=FIRST(A)
      T=TAIL(A)
2     IF (N.EQ.0) GO TO 5
      C=CPROD(P,N,FIRST(T))
      N=N-1
      IF (B.NE.0) GO TO 3
      IF (C.EQ.0) GO TO 4
      B=PFA(N,0)
3     B=PFA(C,B)
4     T=TAIL(T)
      GO TO 2
5     CPDRV=INV(B)
      RETURN
      END
```

```
      INTEGER FUNCTION CPEGCD(P,A,B)
      INTEGER BORROW,FIRST,TAIL,PFA,PFL
      INTEGER CSUM,CDIF,CPROD,CPDIF,CPPROD,CPQREM,CSPROD
      INTEGER A,A1,A2,A3,B,C,D,P,Q,S,I,X,X1,X2,X3,Y,Y1,Y2,Y3,
1     A1=BORROW(A)
      A2=BORROW(B)
      X1=PFA(0,PFA(1,0))
      X2=0
      Y1=0
      Y2=BORROW(X1)
      C=1
      S=0
2     L=CPQREM(P,A1,A2)
      CALL DECAP(Q,L)
      CALL DECAP(A3,L)
      IF (A3.NE.0) GO TO 3
      CALL ERLA(Q)
      CPEGCD=0
      GO TO 10
3     N=CPROD(2,FIRST(A1),FIRST(A2))
      S=CSUM(2,S,N)
      D=FIRST(TAIL(A2))
      N=FIRST(A1)-FIRST(A3)
      DO 12 I=1,N
12    C=CPROD(P,C,D)
4     T=CPPROD(P,X2,Q)
      X3=CPDIF(P,X1,T)
      CALL ERLA(T)
      CALL ERLA(X1)
      X1=X2
      X2=X3
5     T=CPPROD(P,Y2,Q)
      Y3=CPDIF(P,Y1,T)
      CALL ERLA(T)
      CALL ERLA(Y1)
      CALL ERLA(Q)
      Y1=Y2
      Y2=Y3
6     CALL ERLA(A1)
      A1=A2
      A2=A3
      IF (FIRST(A2).GT.0) GO TO 2
7     N=FIRST(A1)-1
      IF (N.EQ.0) GO TO 8
      D=FIRST(TAIL(A1))
      DO 13 I=1,N
13    C=CPROD(P,C,D)
8     IF (S.NE.0) C=CDIF(P,0,C)
9     X=CSPROD(P,X2,C,0)
      Y=CSPROD(P,Y2,C,0)
      W=CPROD(P,FIRST(TAIL(A2)),C)
      CPEGCD=PFL(X,PFL(Y,PFL(W,0)))
10    CALL ERLA(A1)
      CALL ERLA(A2)
```

```
      CALL ERLA(X1)
      CALL ERLA(X2)
      CALL ERLA(Y1)
      CALL ERLA(Y2)
      RETURN
      END


      SUBROUTINE CPERAS(AA)
      INTEGER AA,A,C,N,R
      INTEGER COUNT,TYPE
      A=AA
      R=1
      GO TO 1
 9    RETURN
C   RECURSIVE PROCEDURE CPERAS(A).
 1    IF (A.EQ.0) GO TO 6
      N=COUNT(A)-1
      CALL SCOUNT(N,A)
      IF (N.GT.0) GO TO 6
      CALL DECAP(C,A)
      IF (TYPE(A).NE.0) GO TO 4
      CALL ERLA(A)
      GO TO 6
 4    N=COUNT(A)-1
      CALL SCOUNT(N,A)
      IF (N.GT.0) GO TO 6
      CALL DECAP(C,A)
C   RECURSIVE CALL CPERAS(C).
      CALL STACK2(A,R)
      A=C
      R=2
      GO TO 1
 5    CALL UNSTK2(A,R)
C   END RECURSIVE CALL.
      IF (A.NE.0) GO TO 4
 6    GO TO (9,5), R
C   END RECURSIVE PROCEDURE.
      END


      INTEGER FUNCTION CPEVAL(P,AA,B)
      INTEGER AA,A,B,C,D,E,F,G,P
      INTEGER CINV,CPROD,CPSUM,CSPROD,CSUM,TAIL,TYPE
      A=AA
      C=0
      IF (A.EQ.0) GO TO 5
      A=TAIL(A)
      IF (TYPE(A).EQ.1) GO TO 2
      CALL ADV(C,A)
 1    IF (A.EQ.0) GO TO 5
      CALL ADV(E,A)
```

```
      C=CSUM(P,CPROD(P,C,B),E)
      GO TO 1
2     D=1
      A=CINV(A)
      CALL DLCAP(C,A)
3     IF (A.EQ.0) GO TO 5
      D=CPROD(P,D,B)
      CALL DLCAP(E,A)
      IF (E.EQ.0) GO TO 3
      F=CSPROD(P,E,D,0)
      G=CPSUM(P,C,F)
      CALL CPERAS(E)
      CALL CPERAS(F)
      CALL CPERAS(C)
      C=G
      GO TO 3
5     CPEVAL=C
      RETURN
      END


      INTEGER FUNCTION CPGARN(Q,BB,P,AA,LL)
      INTEGER AA,A,BB,B,C,E,F,G,K,LL,L,M,N,R
      INTEGER BORROW,CGARN,CINV,FIRST,INV,PFA,PFL,TAIL
      IF (LL.NE.0) GO TO 1
      C=CGARN(Q,BB,P,AA)
      GO TO 3
1     A=AA
      B=BB
      L=CINV(LL)
      R=1
      GO TO 4
2     CALL ERASE(L)
3     CPGARN=C
      RETURN
C    RECURSIVE PROCEDURE C=CPGARN(Q,B,P,A,L).
4     C=0
      M=-1
      IF (A.NE.0) CALL ADV(M,A)
      N=-1
      IF (B.EQ.0) GO TO 5
      B=TAIL(B)
      N=FIRST(TAIL(B))
5     IF (A.EQ.0 .AND. B.EQ.0) GO TO 12
      E=0
      F=0
      IF (M.LT.N) GO TO 7
      K=M
      M=M-1
      CALL ADV(E,A)
      IF (K.GT.N) GO TO 8
7     K=N
      CALL ADV(F,B)
```

```
      B=TAIL(B)
      N=-1
      IF (B.NE.0) N=FIRST(TAIL(B))
 8    IF (L.EQ.0 .AND. F.EQ.0) GO TO 5
      IF (TAIL(L).NE.0) GO TO 9
      G=CGARN(Q,F,P,E)
      GO TO 11
C    RECURSIVE CALL G=CPGARN(Q,F,P,E,TAIL(L))
 9    CALL STACK3(A,B,C)
      CALL STACK3(M,N,K)
      CALL STACK2(L,R)
      A=F
      B=F
      L=TAIL(L)
      R=2
      GO TO 4
 10   G=C
      CALL UNSTK2(L,R)
      CALL UNSTK3(M,N,K)
      CALL UNSTK3(A,B,C)
C    END RECURSIVE CALL.
 11   C=PFA(K,PFL(G,C))
      GO TO 5
 12   IF (C.NE.0) C=PFL(BORROW(FIRST(L)),INV(C))
      GO TO (2,10), R
C    END RECURSIVE PROCEDURE.
      END


      INTEGER FUNCTION CPGCD1(P,X,Y)
      INTEGER P,X,Y,A,B,C
      INTEGER BORROW,CPREM,CMONIC
      A=BORROW(X)
      B=BORROW(Y)
 1    C=CPREM(P,A,B)
      CALL ERLA(A)
      A=B
      B=C
      IF (B.NE.0) GO TO 1
      CPGCD1=CMONIC(P,A)
      CALL ERLA(A)
      RETURN
      END


      SUBROUTINE CPINS(A,L)
      INTEGER FIRST,PFL,TAIL
      INTEGER A,L,B,D,M,N
 1    N = L
      D = FIRST(A)
 2    IF (N .EQ. 0) GO TO 3
      IF (FIRST(FIRST(N)) .LE. D) GO TO 3
```

```
      M = N
      N = TAIL(N)
      GO TO 2
3     B = PFL(A,N)
      IF (N .EQ. L) GO TO 4
      CALL SSUCC(B,M)
      RETURN
4     L = B
      RETURN
      END


      INTEGER FUNCTION CPINT(P,A,B,C,DD,R)
      INTEGER A,B,C,DD,D,P,R,T,U,V,W
      INTEGER CDIF,CPDIF,CPEVAL,CPROD,CPSUM,CRECIP,CSPROD,INV,PF
      D=DD
      T=CPEVAL(P,A,B)
      V=CPEVAL(P,D,B)
      V=CRECIP(P,V)
      IF (R.GT.1) GO TO 1
      U=CDIF(P,C,T)
      W=CPROD(P,V,U)
      V=CSPROD(P,D,W,0)
      GO TO 3
1     U=CPDIF(P,C,T)
      CALL CPERAS(T)
      W=CSPROD(P,U,V,0)
      CALL CPERAS(U)
      CALL ADV(V,D)
      V=PFA(V,0)
2     CALL ADV(U,D)
      V=PFL(CSPROD(P,W,U,0),V)
      IF (D.NE.0) GO TO 2
      V=INV(V)
      CALL CPERAS(W)
3     CPINT=CPSUM(P,V,A)
      CALL CPERAS(V)
      RETURN
      END


      INTEGER FUNCTION CPMOD(P,AA)
      INTEGER AA,A,B,C,E,M,N,P,R,T
      INTEGER CMOD,FIRST,INV,PFA,PFL,TAIL,TYPE
      A=AA
      B=0
      R=1
      IF (A.NE.0) GO TO 3
2     CPMOD=B
      RETURN
C     RECURSIVE PROCEDURE B=CPMOD(P,A).
3     IF (TYPE(A).EQ.1) GO TO 4
```

```
      B=CMOD(P,A)
      GO TO 13
 4    B=0
      A=TAIL(A)
      N=FIRST(TAIL(A))
      T=TYPE(FIRST(A))
      M=N
 5    IF (A.EQ.0) GO TO 6
      IF (M.EQ.FIRST(TAIL(A))) GO TO 7
 6    C=0
      GO TO 10
 7    E=FIRST(A)
      A=TAIL(TAIL(A))
C   RECURSIVE CALL C=CPMOD(P,E).
      CALL STACK3(A,B,M)
      CALL STACK3(N,R,T)
      A=E
      R=2
      GO TO 3
 9    C=B
      CALL UNSTK3(N,R,T)
      CALL UNSTK3(A,B,M)
C   END RECURSIVE CALL.
 10   IF (T.EQ.0) B=PFA(C,B)
      IF (T.NE.0) B=PFL(C,B)
      M=M-1
      IF (M.GE.0) GO TO 5
      B=INV(B)
 11   IF (FIRST(B).NE.0) GO TO 12
      CALL DECAP(C,B)
      N=N-1
      IF (N.GE.0) GO TO 11
 12   IF (B.NE.0) B=PFA(N,B)
 13   GO TO (2,9), R
C   END RECURSIVE PROCEDURE.
      END



      INTEGER FUNCTION CPNEG(P,AA)
      INTEGER AA,A,B,C,D,G,P,R
      INTEGER CDIF,INV,PFA,PFL,TYPE
      A=AA
      R=1
      GO TO 1
 9    CPNEG=B
      RETURN
C   RECURSIVE PROCEDURE B=CPNEG(P,A).
 1    B=0
      IF (A.EQ.0) GO TO 6
      CALL ADV(D,A)
      B=PFA(D,0)
      IF (TYPE(A).EQ.1) GO TO 3
 2    CALL ADV(C,A)
```

```
      B=PFA(CDIF(P,U,C),B)
      IF (A.NE.0) GO TO 2
      GO TO 5
 3    CALL ADV(C,A)
C    RECURSIVE CALL G=CPNEG(P,C).
      CALL STACK3(A,B,R)
      A=C
      R=2
      GO TO 1
 4    G=P
      CALL UNSTK3(A,B,R)
C    END RECURSIVE CALL.
      B=PFL(G,B)
      IF (A.NE.0) GO TO 3
 5    B=INV(B)
 6    GO TO (9,4), R
C    END RECURSIVE PROCEDURE.
      END


      INTEGER FUNCTION CPOWER(P,A,K)
      INTEGER A,M,N,P,X
      INTEGER CPROD
      N=K
      CPOWER=1
      IF (N .EQ. 0) RETURN
      X=A
1     M=N/2
      IF (M*2 .EQ. N) GO TO 2
      CPOWER=CPROD(P,CPOWER,X)
      IF (M .EQ. 0) RETURN
2     N=M
      X=CPROD(P,X,X)
      GO TO 1
      END



      INTEGER FUNCTION CPPROD(P,AA,BB)
      INTEGER AA,A,BB,B,C,E,F,G,H,I,J,P,R,S,T,U
      INTEGER CPROD,CPSUM,CSPROD,CSUM,FIRST,INV,PFA,PFL,TAIL,TYP
      A=AA
      B=BB
      R=1
      GO TO 1
 15   CPPROD=C
      RETURN
C    RECURSIVE PROCEDURE C=CPPROD(P,A,B).
 1    C=0
      IF (A.EQ.0 .OR. B.EQ.0) GO TO 13
      CALL ADV(U,B)
      IF (TYPE(B).NE.0) GO TO 2
      F=FIRST(B)
      C=CSPROD(P,A,F,U)
```

```
      GO TO 4
 2    T=0
      S=A
      A=TAIL(A)
 3    CALL ADV(E,A)
      F=FIRST(B)
      GO TO 7
 4    T=TAIL(C)
      S=TAIL(A)
 5    B=TAIL(B)
      IF (B.EQ.0) GO TO 13
      T=TAIL(T)
      A=S
      U=T
 6    CALL ADV(E,A)
      F=FIRST(B)
      IF (TYPE(U).NE.0) GO TO 7
      H=CSUM(P,FIRST(U),CPROD(P,E,F))
      GO TO 12
C   RECURSIVE CALL G=CPPROD(P,E,F).
 7    CALL STACK3(A,B,C)
      CALL STACK2(R,S)
      CALL STACK2(T,U)
      A=E
      B=F
      R=2
      GO TO 1
 8    G=C
      CALL UNSTK2(T,U)
      CALL UNSTK2(R,S)
      CALL UNSTK3(A,B,C)
C   END RECURSIVE CALL.
      IF (T.NE.0) GO TO 11
      C=PFL(G,C)
      IF (A.NE.0) GO TO 3
      A=S
      IF (U.EQ.0) GO TO 10
      DO 9 I=1,U
 9    C=PFL(0,C)
 10   C=PFA(U+FIRST(A),INV(C))
      GO TO 4
 11   J=FIRST(U)
      H=CPSUM(P,J,G)
      CALL CPERAS(G)
      CALL CPERAS(J)
 12   CALL ALTER(H,U)
      IF (A.EQ.0) GO TO 5
      U=TAIL(U)
      GO TO 6
 13   GO TO (15,8), R
C   END RECURSIVE PROCEDURE.
      END
```

```
      INTEGER FUNCTION CPQREM(P,X,Y)
      INTEGER P,X,Y,A,B,C,CDIF,CPROD,CRECIP,CSPROD,D,E,FIRST,I,INV
      INTEGER J,K,L,M,N,PFA,PFL,QUO,TAIL
      QUO=0
      A=0
      N=FIRST(Y)
      IF (N.EQ.0) GO TO 7
      M=FIRST(X)
      QUO=PFA(M-N,0)
      I=TAIL(X)
    1 A=PFA(FIRST(I),A)
      I=TAIL(I)
      IF (I.NE.0) GO TO 1
      A=INV(A)
      B=TAIL(Y)
      E=CRECIP(P,FIRST(B))
      B=TAIL(B)
    2 C=CPROD(P,FIRST(A),E)
      QUO=PFA(C,QUO)
      CALL ALTER(0,A)
      I=TAIL(A)
      J=B
      L=0
      DO 3 K=1,N
      D=CDIF(P,FIRST(I),CPROD(P,C,FIRST(J)))
      CALL ALTER(D,I)
      I=TAIL(I)
    3 J=TAIL(J)
    4 IF (FIRST(A).NE.0) GO TO 5
      CALL DECAP(D,A)
      M=M-1
      IF (L.EQ.1.AND.M.GE.N) QUO=PFA(0,QUO)
      L=1
      IF (M.GE.0) GO TO 4
      GO TO 6
    5 IF (M.GE.N) GO TO 2
      A=PFA(M,A)
    6 CPQREM=PFL(INV(QUO),PFL(A,0))
      RETURN
    7 QUO=CSPROD(P,X,CRECIP(P,FIRST(TAIL(Y))),0)
      CPQREM=PFL(QUO,PFL(0,0))
      RETURN
      END


      INTEGER FUNCTION CPREAD(P,U)
      INTEGER A,B,P,U,PREAD,CPMOD
      A=PREAD(U)
      IF (A.LT.0) GO TO 1
      B=A
      A=CPMOD(P,B)
      CALL PERASE(B)
1     CPREAD=A
      RETURN
      END
```

```
      INTEGER FUNCTION CPREM(P,X,Y)
      INTEGER P,X,Y,A,B,C,D,E
      INTEGER BORROW,PFA,FIRST,TAIL,INV,CDIF,CPROD,CRECIP
      A = 0
      N = FIRST(Y)
      IF (N .EQ. 0) GO TO 6
      M = FIRST(X)
      IF (M .GE. N) GO TO 7
      A = BORROW(X)
      GO TO 6
7     I = TAIL(X)
1     A = PFA(FIRST(I),A)
      I = TAIL(I)
      IF (I .NE. 0) GO TO 1
      A = INV(A)
      B = TAIL(Y)
      E = CRECIP(P,FIRST(B))
      B = TAIL(B)
2     C = CPROD(P,FIRST(A),E)
      CALL ALTER(0,A)
      I = TAIL(A)
      J = B
      DO 3 K = 1,N
      D = CDIF(P,FIRST(I),CPROD(P,C,FIRST(J)))
      CALL ALTER(D,I)
      I = TAIL(I)
3     J = TAIL(J)
4     IF (FIRST(A) .NE. 0) GO TO 5
      CALL DECAP(D,A)
      M = M-1
      IF (M .GE. 0) GO TO 4
      GO TO 6
5     IF (M .GE. N) GO TO 2
      A = PFA(M,A)
6     CPREM = A
      RETURN
      END
```

```
      INTEGER FUNCTION CPROD(P,X,Y)
      INTEGER P,X,Y,A,B
      A=X
      B=Y
      CALL MPY(A,B)
      CALL QR(A,B,P)
      CPROD=B
      RETURN
      END
```

```
      INTEGER FUNCTION CPSUM(P,AA,BB)
      INTEGER AA,A,BB,B,C,E,F,G,I,L,M,N,P,R,S
      INTEGER BORROW,CSUM,FIRST,INV,PFA,PFL,TYPE
      A=AA
      B=BB
      R=1
      GO TO 2
19    CPSUM=C
      RETURN
C    RECURSIVE PROCEDURE C=CPSUM(P,A,B).
2     IF (A.NE.0) GO TO 3
      C=BORROW(B)
      GO TO 15
3     IF (B.NE.0) GO TO 4
      C=BORROW(A)
      GO TO 15
4     C=0
      CALL ADV(M,A)
      CALL ADV(N,B)
      L=M-N
      IF (L.GE.0) GO TO 5
      M=N
      L=-L
      S=A
      A=B
      B=S
5     IF (TYPE(A).EQ.1) GO TO 8
      IF (L.EQ.0) GO TO 7
      DO 6 I=1,L
      CALL ADV(E,A)
6     C=PFA(E,C)
7     CALL ADV(E,A)
      CALL ADV(F,B)
      C=PFA(CSUM(P,E,F),C)
      IF (A.NE.0) GO TO 7
      GO TO 12
8     IF (L.EQ.0) GO TO 10
      DO 9 I=1,L
      CALL ADV(E,A)
9     C=PFL(BORROW(E),C)
10    CALL ADV(E,A)
      CALL ADV(F,B)
C    RECURSIVE CALL G=CPSUM(P,E,F).
      CALL STACK3(A,B,C)
      CALL STACK2(M,R)
      A=E
      B=F
      R=2
      GO TO 2
11    G=C
      CALL UNSTK2(M,R)
      CALL UNSTK3(A,B,C)
C    END RECURSIVE CALL.
      C=PFL(G,C)
```

```
      IF (A.NE.0) GO TO 10
 12   C=INV(C)
 13   IF (C.EQ.0) GO TO 15
      IF (FIRST(C).NE.0) GO TO 14
      M=M-1
      CALL DECAP(E,C)
      GO TO 13
 14   C=PFA(M,C)
 15   GO TO (19,11), R
C   END RECURSIVE PROCEDURE.
      END

      SUBROUTINE CPTOM(N,L)
      INTEGER FIRST,INV,PFA,PFL,TAIL
      INTEGER C,IL,K,NZ,Q,R
1     R = L
11    Q = FIRST(R)
      CALL DECAP(K,Q)
      NZ = N-1-K
12    IF (NZ .EQ. 0) GO TO 13
      Q = PFA(0,Q)
      NZ = NZ-1
      GO TO 12
13    CALL ALTER(Q,R)
      R = TAIL(R)
      IF (R .NE. 0) GO TO 11
2     IL = INV(L)
      L = 0
21    R = IL
      C = 0
22    Q = FIRST(R)
      IF (Q .EQ. 0) GO TO 23
      CALL ALTER(TAIL(Q),R)
      CALL SSUCC(C,Q)
      C = Q
      R = TAIL(R)
      IF (R .NE. 0) GO TO 22
      L = PFL(C,L)
      GO TO 21
23    CALL ERASE(IL)
      RETURN
      END

      SUBROUTINE CPWRIT(P,U,A,L)
      INTEGER A,B,P,Q,U,PFA,CPGARN
      Q=PFA(1,0)
      B=CPGARN(Q,0,2*P+1,A,L)
      CALL PWRITE(U,B)
      CALL ERLA(Q)
      CALL PERASE(B)
      RETURN
      END
```

```
      INTEGER FUNCTION CRECIP(P,X)
      INTEGER P,X,A1,A2,A3,Y1,Y2,Y3,Q
      A1=P
      A2=X
      Y1=0
      Y2=1
      GO TO 2
1     Q=A1/A2
      A3=A1-A2*Q
      Y3=Y1-Y2*Q
      A1=A2
      A2=A3
      Y1=Y2
      Y2=Y3
2     IF (A2.NE.1) GO TO 1
      IF (Y2.LT.0) Y2=Y2+P
      CRECIP=Y2
      RETURN
      END


      INTEGER FUNCTION CSPROD(P,AA,B,N)
      INTEGER AA,A,B,C,D,E,G,I,N,P,R
      INTEGER CONC,CPROD,INV,PFA,PFL,TAIL,TYPE
      A=AA
      R=1
      IF (B.NE.0) GO TO 1
      C=0
C     RETURN POLYNOMIAL C=CSPROD(P,AA,B,N).
9     IF (C.EQ.0 .OR. N.EQ.0) GO TO 14
      CALL ALTER(N+D,C)
      E=0
      IF (TYPE(TAIL(C)).EQ.1) GO TO 11
      DO 10 I=1,N
10    E=PFA(0,E)
      GO TO 13
11    DO 12 I=1,N
12    E=PFL(0,E)
13    C=CONC(C,E)
14    CSPROD=C
      RETURN
C     RECURSIVE PROCEDURE C=CSPROD(P,A,B,0).
1     C=0
      IF (A.EQ.0) GO TO 6
      CALL ADV(D,A)
      IF (TYPE(A).EQ.1) GO TO 3
2     CALL ADV(E,A)
      C=PFA(CPROD(P,E,B),C)
      IF (A.NE.0) GO TO 2
      GO TO 5
3     CALL ADV(E,A)
C     RECURSIVE CALL G=CSPROD(P,E,B,0).
      CALL STACK2(A,C)
```

```
      CALL STACK2(D,R)
      A=E
      R=2
      GO TO 1
 4    G=C
      CALL UNSTK2(D,R)
      CALL UNSTK2(A,C)
C     END RECURSIVE CALL.
      C=PFL(G,C)
      IF (A.NE.0) GO TO 3
 5    C=PFA(D,INV(C))
 6    GO TO (9,4), R
C     END RECURSIVE PROCEDURE.
      END


      INTEGER FUNCTION CSUM(P,X,Y)
      INTEGER P,X,Y
      CSUM=X+Y
      IF (CSUM.GE.P) CSUM=CSUM-P
      RETURN
      END


      INTEGER FUNCTION CVPROD(P,A,B)
      INTEGER CPROD,CSUM,FIRST,TAIL
      INTEGER P,A,B,Q,R,SUM
 1    SUM = 0
      Q = A
      R = B
 2    IF (Q .EQ. 0 .OR. R .EQ. 0) GO TO 3
      SUM = CSUM(P,SUM,CPROD(P,FIRST(Q),FIRST(R)))
      Q = TAIL(Q)
      R = TAIL(R)
      GO TO 2
 3    CVPROD = SUM
      RETURN
      END


      INTEGER FUNCTION GENPR(A,K,M)
      DIMENSION A(K)
      INTEGER A,M,N,D,R,I,PFA
      N=M+2*K-2
      DO 1 I=1,K
 1    A(I)=1
      D=3
 2    IF (D.GT.N/D) GO TO 7
      R=MOD(M,D)
      I=1
      IF (R.EQ.0) GO TO 6
```

```
      IF (MOD(R,2).EQ.0) GO TO 3
      I=I+(D-R)/2
      GO TO 6
3     I=I+D-R/2
6     IF (M.LE.D) I=I+D
      GO TO 4
5     A(I)=0
      I=I+D
4     IF (I.LE.K) GO TO 5
      R=MOD(D,6)
      D=D+2
      IF (R.EQ.1) D=D+2
      GO TO 2
7     GENPR=0
      I=K
8     IF (A(I).EQ.1) GENPR=PFA(M+2*I-2,GENPR)
      I=I-1
      IF (I.GT.0) GO TO 8
      RETURN
      END
```