N O T E S

on the

G E N I E   C O M P I L E R

for the

R I C E   U N I V E R S I T Y   C O M P U T E R

April, 1963

The unit of definition for the Genie compiler is the program, which has the form

```
          DEFINE
             declarations of external variables
               and non-scalar parameters
             function specifications
          PROG(PARAM).= SEQ
             declarations of internal variables
             constant specifications
             command sequence for the calculation
 END
          DEFINE
 LEAVE
|cr stop |1st tab stop
```

Typing of the program is begun by the sequence 'cr tab uc DEFINE'. This 'DEFINE' insures that the compiler does not retain any symbols mentioned by another user of the system. Each line of a program should be begun with a case punch (uc or lc) and is ended by a carriage return (cr). If a statement is so long that it needs to be broken in typing, the sequence 'cr tab tab tab' provides continuation of the statement onto the next line. 'PROG' designates the program name. 'PARAM' designates the parameters of the program, a non-empty list of names separated by commas. The operator '.=' followed by the symbol 'SEQ' signals initiation of code generation for the program. 'END', typed at the left hand margin and followed immediately by a 'cr', terminates the command sequence and initiates final output. Then 'DEFINE' causes the symbol table limit to be backed up so that the compiler retains only its vocabulary symbols, and all external variables backed over are printed out. 'LEAVE', typed at the left hand margin and followed by 'cr cr', causes exit from the system.

Private names, those invented by a user of the Genie compiler, should be formed by the following rules:

1) a single lower case Roman letter;

or 2) an upper case Roman letter, followed by upper case Roman letters, followed by lower case Roman letters, followed by numerals (no spaces intervening).

By rule 1) the following are examples of names:

a   i   p   x

By rule 2) the following are examples of names:

A   CAT   Fn   DDxy   I2   PQ29   Dog3

Concatenation of names implies multiplication of the variables specified. The following are not names:

ab   A B38   Pt4p   M5ef   w10

and will be interpreted respectively as:

a×b   A×B38   Pt4×p   M5×e×f   w×10

In scanning from left to right to collect the characters which comprise a name, the appearance of a character which cannot be concatenated by rule 2) or a space will terminate the collection. Any number of characters may be used in a name, but only <u>five</u> will be retained by the compiler. If lower case Roman letters are imbedded in a name, the first is tallied as two characters. The names

m   Man

will be printed and stored internally as

.M   M.AN

Names in the vocabulary of the compiler may not be used by the coder. These are:

Z	B1	X	REPEA	MATRI	ATAN	VSPAC
R	B2	I	READ	SCALA	TAN	MSPAC
S	B3	LET	PRINT	CONTR	COT	WAIT
T4	B4	EXECU	PUNCH	SIN	LENGT	if
T5	B5	FALSE	DATA	COS	ROW	and
T6	B6	TRUE	INTEG	SQR	COL	or
T7	PF	END	FUNCT	EXP	EOV	not
CC	U	FOR	VECTO	LOG	NEO	FIX
				BCD	INV	TRAN

In any program, variables fall into one of three categories: internal, external or parameters.

Internal variables must be scalars (integers or floating point numbers), and these are assigned storage within the program. Internal variables do not retain their names after compilation; hence, the same name may be used in more than one program with different meanings in each of the programs. Labels on statements are also internal variables.

External variables may be either scalar or non-scalar (programs, vectors, matrices), and all non-scalars must be external. An external variable, at the time the program is run, has its name on the symbol table (ST,*113) and its scalar value or non-scalar codeword in the corresponding value table (VT,*122) entry. External variables of any one program are the common property of all programs in the machine at running time, and the names must have unique meaning throughout the system. All external variables must appear in the program before the 'SEQ'.

Parameters may be either scalar or non-scalar. If they are non-scalar they must be so declared before the 'SEQ'. Parameters are neither internal nor external in the program in which they appear, but will fall into one of these categories somewhere in the sequence of programs which have been transferred to and not exited from in the course of arriving at the given program while running. The names of parameters are not retained while running but are dummy labels that are used only in compilation.

The forms permissible for declarations are illustrated by:

	VECTOR	A
	VECTOR	A, B, C
	VECTORS	A, B, C
|cr	|1st tab	|2nd tab

Before the 'SEQ' all external variables and those parameters which are not floating point scalars must have their types specified. Declarations for use in this area are:

INTEGER	for integers
SCALAR	for floating point scalars
VECTOR	for vectors
MATRIX	for matrices
FUNCTION	for program names not in the vocabulary of the compiler

Parameters that are not declared are assumed to be floating point scalars. As shown above, the plural declaration may be used, and a single name or a list of names may appear at the second tab position.

Internal variables are scalars, either integers or floating point numbers. All those which are not declared are assumed to be floating point, so the only declaration meaningful for internal variables is

INTEGER

The first appearance of an internal integer must be in the INTEGER declaration.

A <u>function</u> is a program which may be referred to in the Genie language, either for <u>implicit execution</u> as 'F' in the command

y=a+F(P)+b

or for <u>explicit execution</u> as 'G' in the command

EXECUTE G(Q)

Implicit execution is meaningful only if the function is single valued. In this case its output is not specified in the parameter list; scalar output must be stored in T7 and U upon exit, and vector or matrix output must have its codeword stored at machine address 240 upon exit. In all other cases explicit execution is required.

The <u>parameters</u> of a function are given as an ordered list of those quantities which are supplied to the function routine by the program which causes it to be executed. In a Genie program a function name must appear with parameters following, as SIN X2 or CALC(q) or MAP(g,VAR), except in declarations. This means that function names may not be used as parameters of other functions.

Functions of one parameter demand special discussion. If the one parameter of a function is a scalar, its <u>value</u> must be provided in T7 upon entry. A single scalar parameter may <u>not</u> be the <u>output</u> of the function. If the one parameter of a function is a vector or a matrix, it may be used for input or output or both; upon entry T7 must contain the codeword address of the vector or matrix with an indirect addressing (*) bit.

Functions of more than one parameter receive their parameters on the push-down storage list addressed by index register B6. If the function has n parameters, the first will be at B6-n on entry, the second at B6-n+1, ..., the n^{th} at B6-1. A scalar is specified in this list by its address, and a vector or matrix is specified by a * bit and its codeword address. Upon exit B6 must be stepped back over the parameters, i.e.,

exit (B6) = entry (B6) - n

for a function of n parameters, n>1.

A function always returns to the path finder setting on entry, i.e., to the instruction following the transfer to it. All fast registers used in functions are saved. The function program must be written to run with the storage exchange (STEX) system activated.

If a function is to be executed implicitly and its output is not floating point, then it must be declared both as a function and as an integer or vector or matrix in a program which uses it. Thus, the function with its parameters is an operand which must be assigned the type of its output if it is to appear within an arithmetic expression.

Every Genie program is a function. It may be used as such by any other Genie program but it may not use itself. In writing Genie programs and in the use of them in other Genie programs care must be taken that parameters are always listed in the same order and that types of parameters are the same at each occurrence.

If a function is sufficiently simple to be defined in <u>one</u> statement, it may be specified within another program. This is done before the 'SEQ' and is illustrated by the definition of f in the statement

$$f(x,y) = 3ax+a^2y, \; a = 2+x$$

|cr |1st tab

The function of f may then be used within the command sequence of the program, as in the command

$$h = k^2 f(m,n)$$

where the closed subroutine f will be applied to the parameters m and n. During compilation, output for f will be produced independent of that for the program in which it is specified. The function is external to the program and may be used by other programs as well at running time since its name will appear on the symbol table.

There is a collection of function names which is known to Genie. These names need not be declared as functions.

NAME	CODEWORD ADDRESS	DESCRIPTION
* * * for implicit execution only * * *		
SIN(A)	200	
COS(A)	201	
SQR(A)	202	
EXP(A)	203	'A' floating point scalar input;
LOG(A)	204	result floating point scalar
ATAN(A)	205	
TAN(A)	206	
COT(A)	207	
LENGTH(A)	210	'A' vector; result integer length of A.
ROW(A)	210	'A' matrix; result integer number of rows in A
COL(A)	211	'A' matrix; result integer number of cols in A
FIX(A)	217	'A' floating point; result integer nearest to A upon rounding up
INV(A)	224	'A' matrix; result matrix which is inverse of A
TRAN(A)	225	'A' matrix; result matrix which is transpose of A
* * * for explicit execution only * * *		
†VSPACE(A,B)	213	'A' vector, 'B' integer; takes space for A of length B
†MSPACE(A,B)	214	'A' matrix, 'B' integer; 'C' integer; takes space for A,B rows by C cols
†CONTROL(n,WXYZ,r,f)	230	'n' integer,'WXYZ' octal,'r' octal or integer,'f' octal or name; control word is composed and *126 in M-SPIREL is executed

†M-SPIREL monitoring on the printer is provided if sense light 14 is off.

Constants of a program may be numerically specified by a 'LET' statement appearing between the 'SEQ' and the 'END'. The statement must be given before the name of the constant is used in the commands of the calculation. The form of this statement is illustrated by:

```
          LET          PI=3.14159
|cr       |1st tab     |2nd tab
```

This is a message to the compiler which causes the floating point number 3.14159 to be used in the program each time the internal variable name 'PI' appears. A 'LET' statement causes no code to be generated. An internal integer may be specified if it has first been appropriately declared, as

```
          LET          K=3
```

An octal configuration may be specified, but it should not be declared as an integer, as

```
          LET          MASK = +777777077
```

where the + inflection concatenated immediately to the left of a number denotes octal conversion of the number. A fixed codeword address may be specified, as

```
          LET          #CDWD = +265
```

so that the codeword for the function, vector, or matrix named CDWD will be addressed at machine address 265 instead of in the value table.

The values of non-scalars may not be specified in a 'LET' statement.

More than one constant may be specified in a 'LET' statement, if they are separated by commas, as

```
          LET          A=3, z=5.41, #PROG = +247
```

All statements of a program between the 'SEQ' and the 'END', except 'LET's and declarations, cause code to be generated. Such statements are called <u>commands</u>. The occurrence of a label on a command causes a <u>command sequence</u> to be initiated. And the ordered set of all command sequences of the program is called the <u>command sequence for the calculation</u>. Each command falls into one of three categories: arithmetic, control, or input-output. These will be discussed in separate sections.

At present at least one command in the program must be given a label. A label is typed at the left hand margin, as 'CALC' in the command

```
CALC              A=B²+B+3.2, B=W+5.1
|cr               |1st tab
```

The form of a <u>simple arithmetic command</u> is illustrated by:

A = arithmetic expression

|cr |1st tab

The form of a <u>compound arithmetic command</u> is illustrated by:

A = arithmetic expression, B = arithmetic expression,...

where more than one equation appears in the command. If there are no interdependencies among the equations of a command, the equations are coded by Genie in the order given. If there are interdependencies, preference will be given to coding the equations from right to left; but if the i^{th} equation depends on the j^{th} equation and $i<j$ (in counting from left to right), then the j^{th} equation will be coded before the i^{th}. So the second and following equations may well be used to define subexpressions of the first (or primary) equation, producing code that will run more efficiently and copy that will be more readable. An example in which reordering will take place is

y=a+b, a=5c/d, b=6, c=b+4

|cr |1st tab

The code generated will evaluate b, then c, then a, then y. On the other hand, the equations in

M=P+Q, a=3, i=j+1

will be coded in the order given.

An <u>operand</u> in Genie is a single variable, a function name followed by a list of arguments in parentheses, or an expression enclosed in parentheses which dictate order of computation in the conventional manner. Order is also implied by relative <u>rank of operations</u>. In order of decreasing rank, i.e., the most binding first, the arithmetic operations are:

unary inflections: - and |...|

subscription

exponentiation

X and /

+ and binary -

Operations that are permitted within an arithmetic expression on the right hand side of an equation are:

1) +, -, X, / between two scalar operands.

   If the operands are both integer or both floating point, the result will be of the same type. If the scalar operands are of different types, the integer will be floated before the operation is carried out, and the result will be floating point.

2) +, -, X, between two non-scalar operands containing floating point elements.

   Standard conventions apply as to restrictions on dimensional compatibility, and the operands must have base indices equal to one. Addition or subtraction of two vectors or two matrices yields a vector or a matrix respectively. Multiplication of two matrices yields a matrix. Multiplication of a vector and a matrix yields a vector. And multiplication of two vectors yields the scalar product which is a scalar.

3) Implied multiplication between operands which appear immediately next to one another, not separated by an operation. The same rules apply as for the explicit X.

4) Exponentiation between two scalars.

   If either or both of the operands is floating point, the result will be floating point. If both of the operands are integers, the result is an integer, zero if the exponent has a negative value. Note that A^B is typed 'A sup B sub', using the superscript and subscript keys on the flexowriter. The counter associated with these carriage moving keys should be set to zero before starting a program and must return to zero at the end of each command.

5) Subscripting of a vector by an integer or of a matrix by a pair of integers separated by commas.

   The result of the operation is assumed to be floating point unless the non-scalar has been declared as both a vector or matrix and as, say, an integer. The expression A_B is typed 'A sub B sup' and return to zero carriage level must be observed as for exponentiation.

6) Unary - applied to a scalar operand.
   The complement of the integer or floating point operand is formed before this operand is combined with any other across a binary operation. This rule is unambiguous but leads to a possibly unexpected interpretation in the case of $(-A^B)$. Code is generated to form $((-A)^B)$. Inflection of the expression A^B should be written $(-(A^B))$.
7) Absolute value of a scalar operand.
   This inflection is denoted by absolute value bar | before and after the operand. These bars are simply parentheses that cause the quantity inside to be taken with positive sign.

The variable on the left hand side of an equation may be a scalar, or a non-scalar, or a subscripted non-scalar (denoting a scalar element of a vector or matrix). All left hand side variables in a command <u>must</u> be distinct, no scalar or non-scalar defined more than once and not more than one element of any vector or matrix defined in any one command.

The '=' joining left hand side to right hand side of an equation causes storage of the computed right hand side into the location or array specified on the left hand side. Compatibility of types is checked for at time of compilation, and an error message is printed out if incompatibility of the two sides is detected. In every case the right hand side dominates and will be stored as calculated, no conversion taking place. A non-scalar on the left hand side must have base indices one. If the right hand side is non-scalar, the storage addressed by the codeword on the left hand side is freed through STEX, the storage control routine in SPIREL, before the store across the '=' takes place.

Genie has the ability to apply the commutative laws of arithmetic to reorder the terms of an expression to provide calculation using a minimum number of temporary stores. In the coding for a scalar expression, the compiler may use the fast T-registers of the computer for temporary storage. Push-down storage addressed by index register B6 is also used for this purpose. When profitable, the T-registers are used by the compiler for scalar variables that are referred to often in an equation. The codeword at machine address 240 is used in the code by the compiler as an accumulator for vectors and matrices produced in the course of evaluating the right hand side of a non-scalar equation. This address may not be used by a coder. Temporary storage for non-scalars is always on the B6-list.

A simple arithmetic command may be of conditional form, as illustrated by

$$A = E_1 \text{ if } P_1,\ E_2 \text{ if } P_2,\ \ldots,\ E_n \text{ if } P_n,\ E_{n+1}$$

|cr |1st tab

where the E_i are arithmetic expressions and the P_i are arithmetic predicates. The code that is generated will evaluate A as E_i for the least i for which P_i is true. If no P_i is true, for $i = 1, 2, \ldots, n$, then A is evaluated as E_{n+1}. E_{n+1} may be omitted from the command, in which case A is not evaluated if all predicates are false. A simple arithmetic predicate is of the form L r R, where L and R are arithmetic expressions and r is a relation, one of $=$, $\neq$, $<$, $\nless$, $\leq$, $\nleq$. A compound arithmetic predicate is formed by joining simple predicates with the operations 'and' and 'or', as in

$$A = 1.0 \text{ if } (B \leq C \text{ or } |C+D| \neq 3.72) \text{ and } D < m+p,\ 2.0 \text{ if } x < 0.0,\ 3.0$$

|cr |1st tab |2nd tab |3rd tab

The 'and' operation is more binding than the 'or'. Parentheses may be used, as in the above example, to dictate computational order.

The form E_1 r E_2 r' E_3 is tempting but not permitted. An equivalent permissible form is

$$E_1 \text{ r } E_2 \text{ or } E_2 \text{ r' } E_3$$

Genie requires a precise sequence of typed characters for the negated relations:

$\neq$ is typed ' = backspace uc | '

$\nless$ is typed ' < backspace uc | '

$\nleq$ is typed ' ≤ backspace | '

Two arithmetic predicates that are exceptions to the form described are 'EOV', asking if the exponent overflow light is on, and its negation 'NEO'. Both of these tests turn the light in the indicator register off.

A conditional arithmetic equation must stand alone as a command. It may not be grouped with other equations in a compound arithmetic command.

Code is generated so that the commands of the program are normally executed in the order written. An explicit variation in this order is indicated by a transfer command, illustrated by

```
          CC = #LOOP
|cr       |1st tab
```

Here 'CC' is the mnemonic for the control counter which is normally stepped sequentially through the orders of the code. 'LOOP' is a label on a command of the program, the command to which control will be passed by this transfer command. The inflection '#' is required in this context to indicate that the address corresponding to LOOP, and not the contents of the location whose address is LOOP, is to be calculated on the right hand side. The '#' inflection is analagous to the 'a' bit in AP1.

The conditional transfer command provides variation in the order of command execution depending upon the truth values of arithmetic predicates. The form of this type of control command is shown by

$$CC = \#A_1 \text{ if } P_1,\ \#A_2 \text{ if } P_2,\ \ldots,\ \#A_n \text{ if } P_n,\ \#A_{n+1}$$

where the A_i are labels within the program and the P_i are arithmetic predicates. The code generated causes CC to be evaluated as the first $\#A_i$ for which P_i is true. If no P_i, for $i=1, 2, \ldots, n$, is true, CC is evaluated as $\#A_{n+1}$. The term $\#A_{n+1}$ may be omitted from the command, in which case CC is unchanged if all P_i are false, so that no transfer is made. The predicates P_i are of the form described in the section on conditional arithmetic commands.

Loops may be realized in Genie language by a combination of arithmetic commands and transfer control commands. A concise notation for a popular loop structure is provided by the loop control commands. The commands of a loop are parenthesized by the FOR and REPEAT commands of the form

```
           FOR        P = A, B, C
                          commands of the loop
           REPEAT
|cr        |1st tab   |2nd tab
```

The parameter of the iteration is P. The initial value of P is given by A, which may be a constant, a single variable, or an arithmetic expression. The positive increment by which P is stepped at the end of each iteration is given by B, which may be a positive constant, a variable which takes on a positive value at the time the 'FOR' is encountered in execution, or an arithmetic expression which will take on a positive value when evaluated. The final value of P is given by C, and the loop will be traversed until P exceeds C in numerical value. The elements of the FOR command must be scalars, either integers or floating point numbers. A 'REPEAT', followed immediately by a carriage return, must be written for every 'FOR'.

Loops may be nested to any level, but distinct iteration parameters must be used at each level within a nest. Transfer of control may be made from a command within a loop to another command within the loop or to a command outside the loop. Transfer from outside a loop to the FOR command is permitted, but transfer from outside a loop to a command within a loop is not permitted. The 'REPEAT' is considered to be within the loop which it terminates; the 'FOR' is not. Any 'FOR' or 'REPEAT' may be labelled for purpose of transfer to it. If addressed from outside the loop, the iteration parameter will have the value it had upon exit from the loop.

The code generated by the compiler when a FOR command is encountered accomplishes the following:

1) iteration parameter = initial value
2) transfer to command beyond corresponding 'REPEAT' if current value of iteration parameter is greater than final value, otherwise proceed to commands of the loop.

The code generated when a REPEAT command is encountered accomplishes the following:

1) iteration parameter = current value of iteration parameter + increment, as specified in corresponding FOR command
2) transfer to step 2) of FOR sequence described above.

The compiler generates the label '←FORn' on each FOR command and '←RPTn' on the corresponding REPEAT command, n = 1, 2, ..., 9, a, b, ... in each program. A coder's label will be used instead if it appears. Thus, FOR and REPEAT commands begin command sequences whether or not they are labelled by the coder.

The command

```
          EXECUTE        PROG(PARAM)
|cr       |1st tab       |2nd tab
```

causes control to be transferred to the program whose name is denoted by 'PROG' in this illustration. 'PROG' must have been declared as a function outside the command sequence for the calculation. 'PARAM' denotes a list of one or more parameters separated by commas. Control is returned from PROG to the next command in the sequence. The interpretation given to the EXECUTE command by Genie is parallel to that for the arithmetic command, the information to the right of the second tab position corresponding to that after the first '=' in an arithmetic command. Thus, a <u>simple conditional EXECUTE command</u> is allowed, such as

```
          EXECUTE        A(P) if a < b + c, B(Q)
```

And a <u>compound unconditional EXECUTE command</u> is allowed, such as

```
          EXECUTE        SUM(x,y), x = 2a/b, y = ab, b = 4
```

The input-output commands are:

	DATA	LIST
	PRINT	LIST
	PUNCH	LIST
	READ	LIST
|cr	|1st tab	|2nd tab

where 'LIST' denotes a collection of names, not expressions or names assigned machine addresses in 'LET' statements, of scalars or non-scalars with base indices equal to one. Function (or program) names may not appear in the argument list of an input-output command. Neither may vector or matrix elements in the subscript notation be designated in such an argument list.

The DATA command provides reading of manually punched signed decimal numbers from paper tape. The list given in the command may contain any type of variable. If a decimal point appears, the number will be converted to floating point within the machine; the absence of a decimal point causes conversion to integer form. Every number must be followed by a carriage return, tab, or comma. Integers greater than or equal to 2^{15} in absolute value are meaningless; floating point significance to more than fifteen places is not meaningful. A floating point number may be followed by the sequence 'e signed integer' which will cause it to be multiplied by 10 to the signed integer power upon conversion. The absence of a sign on a number implies positive sign. Then

punched	328 cr	converts to	integer 328
	46.9cr		floating point 46.9
	.469e2cr		floating point 46.9
	-5391cr		integer -5391
	-69.e-1cr		floating point -6.9

Scalars must be punched as single numbers in the format described. A vector of length n is punched as the sequence of n+1 numbers: integer n, first element, ..., n^{th} element. A matrix of m rows by n columns is punched as the sequence of mn+2 numbers: integer m, integer n, element (1,1), element (1,2), ..., element (1,n), element (2,1), ..., element (2,n), ..., element (m,1), ..., element (m,n). When the DATA command is executed, the proper tape is assumed to be in the reader. If sense light 14 is off, the line

DATA NAME

will be printed out for each quantity read, where 'NAME' is as designated in the program containing the READ command. Thus, printer monitoring of 'DATA' applied to parameters bears the dummy parameter name, not the name of the argument supplied as the parameter.

The PRINT command provides output on the fast line printer of any scalar or non-scalar quantities. These are labelled by the name given in the routine in which the PRINT command appears. Scalars are printed one per line. Vectors are printed five elements per line. Matrices are printed by row, five elements per line.

The PUNCH command and the READ command may be applied only to variables which are named on the symbol table at the time the command is executed. All external variables of the program in which the 'PUNCH' appears and those parameters which at the time of execution are indeed external in some higher level program fall into this category. Care must be taken to apply these commands properly to parameters as there are no checks built into the compiler or input-output program to insure presence on a particular name on the symbol table. 'PUNCH' provides, for each variable listed, a single control word, followed by the name as it appears on the symbol table, followed by the data in hexad with checksum. For a scalar the SPIREL control word has wxyz=0040; for a vector the control word has wxyz=0240; for a matrix the control word has wxyz=0440. These output paper tapes may be loaded through SPIREL symbolically or they may be read with a READ command. In fact, only tpaes of the form produced by a PUNCH command may be read by a READ command.

Additional forms of input and output may be obtained by use of SPIREL programs directly, but those provided by the input-output commands should be sufficient for a large number of problems.

In the Genie language the index registers of the computer may not be mentioned, but limited use may be made of the full-length T-registers.

T7 is used for output of a scalar from a single valued function that will be executed implicitly. The last command in the program should be of the form

```
               T7= calculated output
|cr           |1st tab
```

T6, T5, and T4 may be used as the names of scalar variables computed in other than the first equation of an arithmetic command. Genie will not make use of any T-register mentioned by the coder, and code efficiency may be increased by explicit assignment of auxiliary variables to these fast registers. For this purpose only T6, T5, T4 are available, and they should be called upon in this order since Genie will use only Ti for i less than the smallest Tj mentioned by the coder. The command

$$M=T6/T5,\ T6=a+b,\ T5=(c^2+c-4.1)/d$$

is an example of coder use of fast registers.

Values of fast registers other than B6 and PF are not preserved from one Genie language command to another.

The assembly language recognized by Genie is called <u>AP2</u>. Instructions in the AP2 language may interspersed at will with commands in the Genie language within the command sequence for a Genie program.

AP2 conventions differ from those of AP1 in the following respects:

1) The 'a' inflection of AP1 is represented by '#' in AP2.

2) The normal mode for numbers is decimal instead of octal. In AP2 a string of numeric characters preceded by a '+' inflection will be interpreted as an octal number.

3) The pseudo orders of AP1 do not exist in AP2.

Frequent use will probably be made of AP2 language for setting and testing of sense lights since no notation for such operations exists within the Genie language. To turn on sense light 3:

```
                               SLN           +10000
|cr           |1st tab        |2nd tab      |3rd tab
```

To ask if sense light 10 is off:

```
                               IF(SLF)SKP    +00040
                               TRA           PT2       ← to PT2 if SL10 on
                        continuation of calculation    ← to next command
                                                          if SL10 off
```

In AP2 commands, the coder may make use of the fast registers, taking care to preserve the value of PF for reference to parameters and to use B6 for temporary push-down storage only.

Alphabetic information for output on the printer may be defined by the BCD command, as illustrated by

```
 MESS1      BCD          _ _TEMPUS_FUGIT
|cr        |1st tab    |2nd tab
```

where _ indicates a space when typing. The command may continue onto succeeding lines at the 3rd tab position by use of the 'cr tab tab tab' sequence. A space is inserted by Genie between the last character of one line and the first of the next line. At the place such a BCD command appears in the command sequence for the program, the printer code for the information is inserted in the code for the program, nine characters per word. Of course, what is generated is not executable, so transfer around BCD commands must be explicitly coded.

Once alphabetic information has been specified, it may be set into the print matrix at any position on the line, one word (i.e., nine characters) at a time, and then printed with program *127 in SPIREL. An AP2 code sequence for printing MESS1 starting at print position 12 is

```
           PF          RPA         RSPF
           Z           SB3         12,U→B1
                       CLA         MESS1,U→T7
                       TSR         *+127,B1+1
                       CLA         MESS1+1,U→T7
                       TSR         *+127
                       TSR         *+127,B1+1
 RSPF                  SPF         Z
|cr        |1st tab   |2nd tab    |3rd tab
```

Detailed discussion of program *127 may be found in earlier write-ups on SPIREL. For printing MESS1 at the left hand margin, the Genie language command

```
           EXECUTE     CONTROL(2,+4010,0,MESS1)
|cr        |1st tab   |2nd tab
```

with SL14 on will provide the desired output. The parameters in this command indicate that two words starting at the location named MESS1 are to be printed in hexad form. Printing is produced 108 characters per line, as many lines as necessary. In the example 14 characters require two words of storage, hence the value 2 for the first parameter to CONTROL. The function CONTROL is explained in the FUNCTIONS section.

The sizes of code sequences and programs generated by the Genie compiler are limited by the size of the memory. With 8K of memory no code sequence may contain more than 300 (octal) instructions, and the entire program may not exceed 1000 (octal) instructions in length. The compiler does not check for overflow, but it should be apparent at time of compilation if the limits are exceeded. No absolute correspondence can be established between the length of a Genie program in symbolic form and the length of the absolute program it causes the compiler to generate. Roughly, though, a page of Genie language segmented into four command sequences should not exceed the size restrictions imposed on the code generated.

While compiling, the number of private symbols which may be stored is 70 (decimal). While running a system, the standard M-SPIREL allows for 64 external names on the symbol table.

The Genie PLACER system provides for operations on symbolic Genie program tapes. It is located on the MT System magnetic tape at block 101.01. When this PLACER is read into memory program *240 is executed, and the stop

(I): 00 HTR CC

occurs. The set of options to be exercised should then be designated in the sense lights:

SL1 on	read symbolic tape
SL2 on	edit
SL3 on	punch (edited) symbolic tape
SL4 on	list (edited) symbolic tape
SL5 on	check tape punched

After putting the symbolic tape in the reader, pushing CONTINUE causes the specified operations to be carried out in order as described below.

SL1 on, READ. The tape to be read must contain only one symbolic program, this begun with one carriage return and terminated by two carriage return punches. All characters beyond the last cr on the tape are ignored by the system. When the reading is complete, the system has in the machine a tape image.

SL2 on, EDIT. The stop

(I): 02 HTR CC

occurs. The edit tape is placed in the reader. Pushing CONTINUE causes this tape, which must contain only the corrections for the tape image in the machine, to be read. When reading is complete, PLACER's tape image in the machine is edited.

Each correction is specified by three parameters: the initial carriage return number (i), the final carriage return number (f), and the number of lines in the symbolic correction (n). A line in a symbolic program is terminated by a carriage return, these being numbered from 1 on listings. The n lines of a correction will replace the portion of the program read from and not including carriage return i through carriage return f.

Note that n=0 effects a deletion. The last carriage return on a symbolic tape must <u>not</u> be replaced. On a single edit tape f of one correction may not equal i of another correction. The format for punching the correction parameters is:

(l.c.) i (sp) f (sp) n (cr)

<u>SL3 on, PUNCH</u>. The tape image in the machine is punched out on paper tape.

<u>SL4 on, LIST</u>. The tape image in the machine is listed on the fast line printer with carriage return numbers. A lower case Roman letter is printed as '. upper case letter '. Superscripts and subscripts are printed above and below the main line. Unfortunately, '$\ddagger$' prints as '|', the '=' being lost because the two characters are too close to each other on the print wheel.

<u>SL5 on, CHECK</u>. The stop

(I): 05 HTR CC

occurs if the tape to be checked is not in the reader. Pushing CONTINUE causes the tape that is read to be compared to the tape image in the machine. An error print is given if the comparison fails.

The Genie system is on the MT System magnetic tape at block 101.02. When the system is read into the memory, program *240 is executed, causing a page restore on the printer and some feed on the punch. The stop

(I): 00 HTR CC

occurs. The symbolic Genie program tape should then be placed in the reader. No sense lights should be on. Pushing CONTINUE causes the tape to be read. This reading is very erratic as the text is being processed by Genie as it is read. When the statement

```
 END
|cr
```

is read, output of the program is provided on the printer and the punch. The statement

```
          DEFINE
|cr      |1st tab
```

causes printing of the external variables of the program just compiled. Then the statement

```
 LEAVE
|cr
```

causes exit from the compiler to program *240. A page restore and feed are again provided. When the stop

(I): 00 HTR CC

appears, the system is ready to compile the next symbolic Genie program.

An alternative to the 'LEAVE' is the statement

```
          WAIT
|cr      |1st tab
```

where 'WAIT' is followed <u>immediately</u> by two cr punches. When the compiler encounters this statement, the stop

(I): 11 HTR CC

occurs within the compiler. No page restore or feed on the punch is provided. Pushing CONTINUE with another symbolic Genie program tape in the reader results in compilation of this next program. Thus, output for two or more programs may be run together if each but the last is terminated with a WAIT statement and the last is terminated with a LEAVE statement.

SYMBOLIC ADDRESSING IN M-SPIREL

In the Genie language quantities are normally identified by name, not by the machine address where the corresponding value or codeword is located. The M-SPIREL system provides facilities for addressing scalars, parograms, vectors, and matrices by name. When a control word is read from paper tape by the SPIREL communications linkage beginning at location 20, a null f field will cause program *126 (XCWD) to read what follows on paper tape as a 5-hexad name preceded by a cr punch. The name is added to the symbol table (ST,*113) if it is not already present. Then the f field is assigned the address in the value table (VT,*122) which parallels the name in ST. Under program control a control word with null f may be given in T7, a 5-hexad name left justified in T4, and entry made to the second order of *126 with the AP2 order

TSR   *+126, CC+1

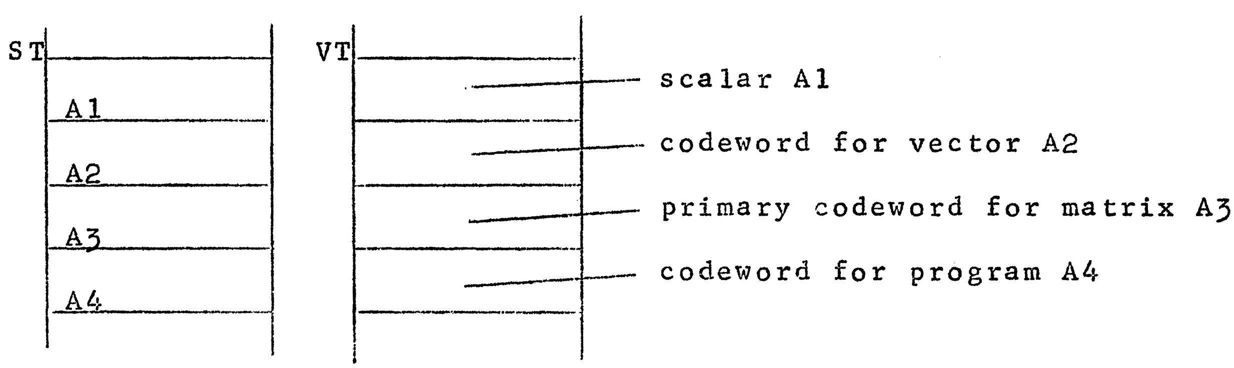
Again, the f field is assigned the appropriate VT address.

The name must be given as 5 printer hexads, as

54-40-55-25-25	for	MAN
54-26-40-55-25	for	Man
54-40-55-01-25	for	MAN1
26-54-25-25-25	for	m

These configurations are not always conveniently punched on the flexowriter since case punches may not appear, the '26' hexad is given by a backspace punch, and the '25' is given by the tab punch.

Given the ST-VT configuration



the control word with symbol

cr 00001-0030-0000-00000 cr 40-01-25-25-25

will cause the scalar A1 in decimal form to be read into A1's VT entry. The control word with symbol

cr 00000-4130-0000-00000 cr 40-02-25-25-25

will cause the vector A2 with codeword in A2's VT entry to be printed in decimal form. The control word with symbol

cr 00000-5440-0000-00000 cr 40-03-25-25-25

will cause the matrix A3 with primary codeword in A3's VT entry to be punched with symbol. The tape punched will load at a later time, creating a matrix with primary codeword in A3's VT entry, even if this entry is not in exactly the same relative VT location. The control word with symbol

cr 00004-0420-0003-00000 cr 40-03-25-25-25

will cause the space currently addressed by the codeword in A3's VT entry to be freed. Then a 4 by 3 matrix of zeroes to be created and addressed by the codeword in A3's VT entry. The control word with symbol

cr 00000-4100-0000-00000 cr 40-04-25-25-25

will cause the program A4 with codeword in A4's VT entry to be printed out in octal. The control word with symbol

cr 00001-4030-0000-00000 cr 40-01-25-25-25

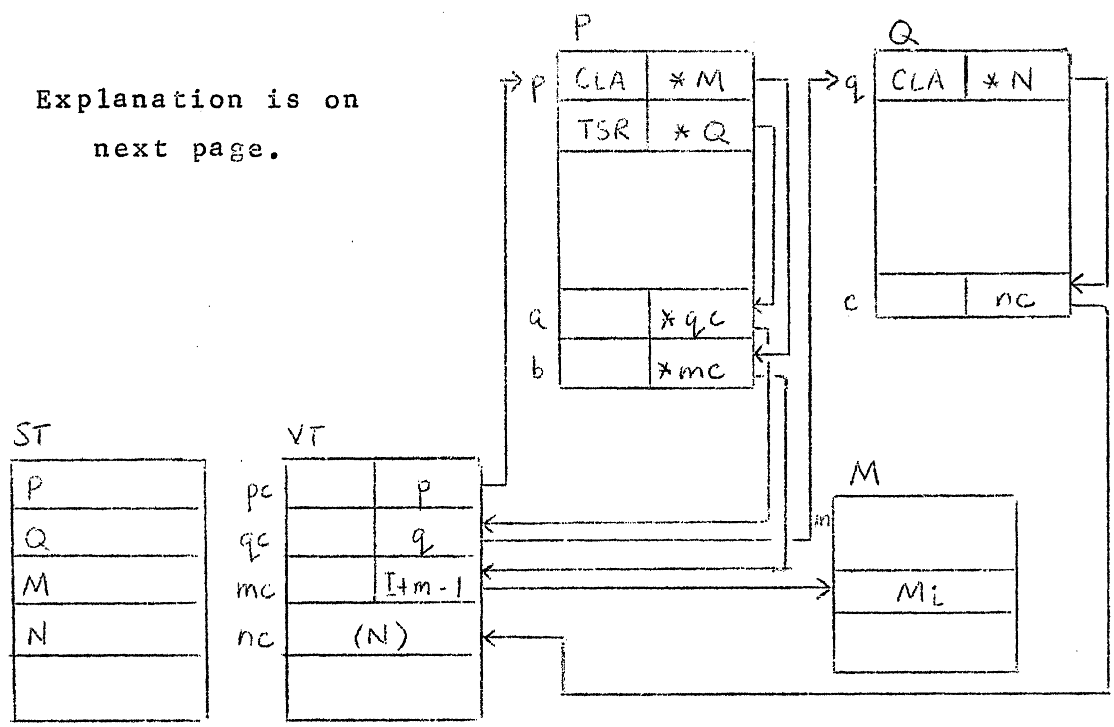
will cause the scalar A1, stored in A1's VT entry, to be printed out in decimal.

SYMBOLIC CROSS REFERENCES

An absolute Genie program, one that has been generated by the compiler, contains one reference word for each external variable referred to in the program. An order which addresses an external variable does so through the reference word with indirect addressing. At execution time the reference word for a scalar contains the value table (VT) address where the scalar is stored; for a non-scalar it contains an indirect addressing (*) bit and the VT address where the codeword is stored. For any Genie program the output tape is in two sections, the program itself in hexads with no checksum which will be loaded symbolically through M-SPIREL, and a control word followed by a list which will load symbolic cross references into the program. This operation supplies proper VT addresses in the reference words of the program.

The figure below illustrates symbolic interconnections between two named programs and the named data to which they refer.

Explanation is on next page.



P and Q are programs, M is a vector, and N is an external scalar. P refers to Q and M through the reference words a and b respectively. Q refers to N through the reference word c. The VT addresses for Q and M are shown as qc and mc respectively, and these are inserted into a and b by loading symbolic cross references into program P. The VT address for N is shown as nc, and this address is inserted into c by loading symbolic cross references into program Q. The paths of addressing from orders of P and Q to the data addressed are shown by arrows in the figure.

Programs written in <u>AP1</u> language and loaded with numeric codeword addresses rather than names may, with some effort, refer to external quantities whose names are in ST with values or codeword in VT. When writing such a program, a block of reference words should be created within the program. For a <u>scalar</u> named SS the reference word should be written

```
 SS                 BCD          SS sp sp sp 0 0 0 0
|cr    |1st tab    |2nd tab     |3rd tab
```

For a <u>program</u> named PP the reference word should be written

```
 PP                 BCD          PP sp sp sp A 0 0 0
```

For a <u>vector</u> named VV the reference word should be written

```
 VV                 BCD          VV sp sp sp A 0 0 0
```

For a <u>matrix</u> named MM the reference word should be written

```
 MM                 BCD          MM sp sp sp A 0 0 0
```

The 'A' in the above BCD instructions provides the * bit required in reference words for non-scalars. Within the code the data is always addressed through the reference words with indirect addressing, as

```
                    FAD          *SS
                    TSR          *PP
                    CLA          *VV
                    STO          *MM
```

Once such an AP1 program is in the machine, proper VT addresses need to be inserted into the address fields of these reference words. Program *173, SXREF, provides a means of filling a block of reference words in the form described above. One "control word" is punched on paper tape for each block of reference words to be operated on by SXREF. The form of this "control word" is

cr nnnnn 0000 rrrr fffff

or

cr nnnnn 0000 rrrr 00000 cr sssss

where nnnnn gives in octal the length of the block of reference words, rrrr gives in octal the relative address within the program of the first word of the block, fffff gives the codeword address of the program if it has been loaded numerically, and sssss gives the 5-hexad name of the program if it has been loaded symbolically. When executed, SXREF will read these "control words" and perform the designated cross referencing until a null word is detected or the end of the paper tape is encountered.

Once a Genie absolute program is read into the machine and its symbolic cross references have been loaded, the program is in a form that is dependent upon the exact contents and order of ST and VT. It may be desirable to punch with name a single program or a system. To reload such tapes, the ST-VT must first exist in the machine precisely as they did at the time the punching took place.

Program *174, CNTXT, provides for punching of a tape which re-establishes <u>context</u>: the value of 117 (current length of ST and VT), correction of *113 (ST) to its current length, clearing of *122 (VT) to its current length. This tape must then be loaded before any items whose name appear on ST as punched. If sense light 13 if off CNTXT proceeds to punch in hexad with checksum all quantities with names in ST for later symbolic loading.

NUMBER TO NAME
CONVERSION

It may be that programs or data which is punched to be loaded at specific addresses or with numbered codeword addresses need to be converted to symbolic loading form for use in a Genie-coded system.

Program *172, SMBLZ, will punch out with the name specified constants loaded into numbered addresses or blocks and arrays loaded with numbered codeword addresses. SMBLZ reads from paper tape the following information about each item to be punched:

cr sssss tab x tab nnn

where sssss is the 5-hexad name which is to be given to the item, x is the digit 0 if the item is a scalar, x is the digit 1 if the item is a program or vector or matrix, and nnn is the three digit address or codeword address where the item is located in memory at the time this punching takes place. If the item is a matrix, all of the array will be punched.

SMBLZ will punch all items described on one tape, exiting only when end of tape is detected. If sense light 13 is on when SMBLZ is executed, tape feed will be supplied between the items punched.

Genie Spirel is located on the MT System magnetic tape at block 101.03. This is a full M-SPIREL and the set of programs which provide support for compiled programs at execution time. The specific contents are listed below.

NAME	CODEWORD ADDRESS	DESCRIPTION
full M-SPIREL		
	* * * utility programs * * *	
SMBLZ	172	see NUMBER TO NAME CONVERSION
SXREF	173	see SYMBOLIC CROSS REFERENCES
CNTXT	174	see CONTEXT OUTPUT
*** programs whose names may be used in Genie language ***		
SIN	200	
COS	201	
SQR	202	
EXP	203	
LOG	204	
ATAN	205	
TAN	206	
COT	207	see FUNCTIONS
LENGTH	210	
ROW	210	
COL	211	
VSPACE	213	
MSPACE	214	
FIX	217	
INV	224	
TRAN	225	
*** programs which may be used by Genie-generated programs ***		
	212	used for DATA, PRINT, PUNCH, READ command
	215	integer to an integer power
	216	floating point number to a floating point power; uses 203, 204
	220	copy of vector or matrix

NAME	CODEWORD ADDRESS	DESCRIPTION
	221	addition of two vectors or two matrices
	222	subtraction of two vectors or two matrices; uses 221
	223	multiplication of vectors or matrices
	226	multiplication of floating point scalar and vector or matrix

Available for use by the coder are addresses 231-237 and 241-277. The system occupies about 6,000 (octal) words of storage and may be cut down by extracting just those programs necessary to a particular system.

The procedure for testing Genie programs should follow an outline similar to the following:

1) load Genie SPIREL from magnetic tape
2) load private programs
3) activate STEX with control word 00000-3120-0000-00135
4) load data items which are prefixed with SPIREL control words
5) position "run tape" which contains the control word

cr 00000-3100-0000-00000 cr PPPPP

where PPPPP is the 5-hexad name of the program to be executed, followed by any data to be read by the program. A "fetch" from location 21 or a CONTINUE to 20 will then cause PPPPP to be executed by SPIREL.

The first version of a Genie program should contain ample PRINT commands that provide display of intermediate results. These may be edited out of the program for production or their execution may be conditional upon sense light settings.

A program should be tested with sense light 14 off. This causes monitoring on the printer of all SPIREL operations, all input-output operations, and all space taking operations. Such information is often a valuable debugging aid.

If a program stops unexpectedly while it is being checked out, the following information may be of value:

- contents of CC, showing in which program the stop occurred
- contents of P2, showing where the last transfer occurred
- contents of PF, showing where the last transfer to subroutine occurred
- a dump of any programs in which values of internal variables may be of interest
- a dump of ST-VT (using SPIREL control word 00000-0500-000-00000), showing values of external scalars and codewords for external non-scalars which are defined at the time
- a dump of any external arrays which may be of interest

Tracing of Genie programs is not advised. But if it is done, care must be taken not to trace transfers to programs 136 (SAVE), 137 (UNSAVE), 212 (INPUT-OUTPUT).

The program SUBR takes two vectors, V1 and V2, and a scalar, SCLR, as input parameters and returns two more vectors, SCNT and VPRIME, as output.

If V1 and V2 are of the same length, their dot product DPROD, is computed and V1 is multiplied by SCLR. If their lengths are different, an indicator is turned on for later testing.

Next, space is taken for the vector SCNT and its elements are evaluated as: $SCNT_j = 0$ if $V1_j$ is within 0.001 of a multiple of $\pi/2$, otherwise $SCNT_j = \sec(V1_j)$.

After SCNT is evaluated, the indicator is tested. If it is off, space is taken for the vector VPRIME and it is evaluated as a function of V2 and SCNT; if the indicator is on, the calculations on VPRIME are skipped.

Finally, the indicator is turned off and the values of SCLR, DPROD, V1, SCNT, and VPRIME are printed.

Notes on Symbolic Listing:

Line	Remark
3,4	All non-scalars, all functions not in the vocabulary of the compiler, and all external scalars must be declared before the SEQ.
5	The one-line definition of function REM is also located before the SEQ; the user must supply a function INT to compute the largest integer contained in a number. External specifications apply to the function REM as well as to the main program.
7	LNG1, LNG2, and j are declared as integers. Since this statement appears after the SEQ, the integers are internal to the program SUBR.
10	HALFPI is defined as 1.570796; this value is used in the code wherever the name appears. Since 'HALFPI' is more than five characters long, it will appear on listings as 'HALFP' and will not be distinguished from any other character beginning 'HALFP'.
11	Several equations separated by commas may appear on one line.

Line	Remark
12	Since the value of CC is to be unchanged if the condition is not satisfied, the alternative value is omitted. Note that the Genie lister prints | for ǂ and ← for #.
13,14	Vector V1 is multiplied by vector V2 for a scalar result and each element of V1 is multiplied by SCLR. The use of × to indicate multiplication on line 13 is synonymous with the juxtaposition of the factors on line 14.
17,21,22	Execution of a function may be called for explicitly with an EXECUTE command or implicitly in an arithmetic command, depending on the function.
20,23,24,27,30	These commands control a loop indexed on j. A test is made at the beginning of each pass through the loop to determine which of two calculations is to be performed for the current value of j. At the end of each calculation, j is incremented and control is transferred to the initial test if j ≤ LNG1 or to the first instruction after the loop if j > LNG1.
34,37	This is a simpler method of loop control; it is useful for loops with positive increments and a single exit point.
35,36	A statement may extend for more than one line. The case punch for the second line <u>follows</u> the third tab in the 'cr tab tab tab' sequence.
16,26,31,32,40	AP2 instructions may be interspersed with GENIE statements; no special indication is necessary. AP2 commands that use SKP, JMP, or otherwise depend on CC should be used with caution; It is difficult to predict the number of machine language instructions which a GENIE command will generate.
42	'END' terminates the command sequence by generating code for return of control to the program at the next higher level.
44	'LEAVE' causes exit from Genie at compilation time.

EPILOGUE

The Genie compiler is the invention of John K. Iliffe, now with Ferranti, Ltd. in London. Major contributions to its realization have been made by Jane G. Jodeit, T.A. Kitchens, Jr., and Jo Kathryn Mann.

Programming development has been supported by the National Science Foundation under grant NSF G-17934. Construction of the Rice University Computer was supported by the Atomic Energy Commission under contract AT-(40-1)-1825, further development under contract AT-(40-1)-2572.

Genie may well be improved and extended by future efforts in a number of areas:

(i) Notation for sense light interrogation would be very useful.

(ii) The case of a program with no command labels should be handled properly.

(iii) Function names should be allowed as parameters.

(iv) The machinery for Boolean variables exists but needs to be checked out and made available.

(v) At compilation time a list of programs referred to in the compiled code should be provided.

(vi) Compound conditional commands should be permitted.

(vii) Checks on overflow of size limits and various other compiler diagnostics should be provided and documented.

(viii) A major effort would be required to allow programs to use themselves, but this might be interesting and worthwhile.

(ix) More elaborate input-output facilities would be useful.

Jane G. Jodeit

Rice University
Houston, Texas

```
SUBR     START NEW PROGRAM.
*BGIN    PROGRAM SEQUENCE.
ARND     PROGRAM SEQUENCE.
EVEN     PROGRAM SEQUENCE.
COMPA    PROGRAM SEQUENCE.
LOW      PROGRAM SEQUENCE.
COMPB    PROGRAM SEQUENCE.
*FOR1    PROGRAM SEQUENCE.
*RPT1    PROGRAM SEQUENCE.
OMIT     PROGRAM SEQUENCE.
```

```
                 START PHASE 2 AND NORMAL OUTPUT SEQUENCE.
SUBR  .=          0   *BGIN      1       1001000024400000136
                  1              2       14000700410077760
                  2              3       12170007020000001        V1
                  3              4       47216410000010000 1
                  4              5       14000000440000210
                  5              6       14000700400000000
                  6              7       12000100400100232        LNG1
                  7             10       12170007020000002        V2
                 10             11       47216410000010000 1
                 11             12       14000000440000210
                 12             13       14000700400000000
                 13             14       12000100400100226        LNG2
                 14             15       12170006000100224        LNG1
                 15             16       60205000000100224        LNG2
                 16             17       10100000400100001
                 17             20       10100000400100002
                 20             21       60251000400000000
                 21             22       10100000400100002
                 22             23       12170000400100034        ARND
                 23             24       12004040000000000
                 24             25       14000100400000240
                 25             26       12170042020000001        V1
                 26             27       47216410000010000 1
                 27             30       14000000440000220
                 30             31       14000700400000000
                 31             32       12170042020000002        V2
                 32             33       47216410000010000 1
                 33             34        4000041440000223
                 34             35       14000700400000000
                 35             36       12000100400100205        DPROD
                 36             37       14000100400000240
                 37             40       12170042020000001        V1
                 40             41       47216410000010000 1
                 41             42       14000000440000220
                 42             43       14000700400000000
                 43             44       15040000060000000        SCLR
                 44             45       47216410000010000 1
                 45             46       24000000440000226
                 46             47       14000700400000000
                 47             50       12170041020000001        V1
                 50             51       47216010000010000 1
                 51             52        4000042440000135
                 52             53       14000700400000000
```

53		54	5040152000000240	
54		55	22000100400200000	
55		56	412164100000400000	
56		57	12170040400100001	EVEN
0	ARND	60	14200000400040000	
0	EVEN	61	12170000020000003	SCNT
1		62	12000126410000000	
2		63	2010000400100156	LNG1
3		64	12000126410000000	
4		65	472164100000100001	
5		66	14000000440000213	
6		67	14000700400000000	
7		70	12170000400000001	
10		71	12000100400100153	.J
0	COMPA	72	12174041000100152	.J
1		73	12174000060000001	VI
2		74	12000171400100151	+P1
3		75	412160326400000000	
4		76	2010000400100150	HALFP
5		77	12000126410000000	
6		100	472164100000100001	
7		101	14000000440100146	REM
10		102	14000700400000000	
11		103	10655000000100145	NUMB
12		104	10100000400100002	
13		105	12170000400100025	LOW
14		106	12004040000000000	
15		107	12174041000100135	.J
16		110	12174007060000001	VI
17		111	472164100000100001	
20		112	14000000440000201	
21		113	14000700400000000	
22		114	11470000000100135	+ONEF
23		115	12000126410000000	
24		116	12170041000100126	.J
25		117	12170066010077776	
26		120	12000100460000003	SCNT
27		121	12174000400000001	
30		122	11000000000100122	.J
31		123	12000100400100121	.J
32		124	12174000000100120	.J
33		125	10251000000100114	LNG1
34		126	10100000400100002	
35		127	12170000400177741	COMPA
36		130	10100000400100001	
37		131	12170000400100015	COMPB
40		132	12004040000000000	
0	LOW	133	12170041000100111	.J
1		134	12170000000100116	NUMB
2		135	12000100460000003	SCNT
3		136	14200000400040000	
4		137	12174000400000001	
5		140	11000000000100104	.J
6		141	12000100400100103	.J
7		142	12174000000100102	.J
10		143	10251000000100076	LNG1
11		144	10100000400100002	
12		145	12170000400177723	COMPA
13		146	12004040000000000	
0	COMPB	147	10207000400040000	
1		150	10100000400100054	OMIT
2		151	12170000020000004	VPRIM
3		152	12000126410000000	
4		153	2010000400100067	LNG2
5		154	12000126410000000	
6		155	472164100000100001	

7		156	14000000440000213	
10		157	14000700400000000	
0	←FOR1	160	10100000400100002	
1		161	0	
2		162	0	
3		163	12170000400000001	
4		164	12000100400100060	.J
5		165	12170000400000001	
6		166	12000100400177771	←FOR1
7		167	12170000000100053	LNG2
10		170	12000100400177770	←FOR1
11		171	12170000000177767	←FOR1
12		172	10211000000100052	.J
13		173	10100000400100031	←RPT1
14		174	12170006000100050	.J
15		175	12174041000000006	
16		176	12174004060000003	SCNT
17		177	12174041000000006	
20		200	12174000060000002	V2
21		201	10251000000000004	
22		202	10100000400100003	
23		203	12174041000000006	
24		204	12174000060000002	V2
25		205	10100000400100010	
26		206	12174041000000006	
27		207	12174000060000002	V2
30		210	210655000000100043	NUMB
31		211	10100000400100002	
32		212	12170000400000000	
33		213	10100000400100002	
34		214	12174041000000006	
35		215	12174000060000001	V1
36		216	12000126410000000	
37		217	62004041000000000	
40		220	12170066010077776	
41		221	12000100460000004	VPRIM
0	←RPT1	222	12170000000177735	←FOR1
1		223	11040100000100021	.J
2		224	10100000000177743	←FOR1
0	OMIT	225	14200400400040000	
1		226	14000100400000002	
2		227	10100000440000212	
3		230	624253612560000000	SCLR
4		231	435761564300100012	DPROD
5		232	650125252520000001	V1
6		233	624255632520000003	SCNT
7		234	655761505420000004	VPRIM
10		235	0	

SUBR SYMBOL TABLE.

112	SCLR	102	0	0	0	0	
113	←BGIN	300	1	3	0	0	
114	LNG1	200	242	3	0	1	
115	LNG2	200	243	3	0	1	
116	.J	200	245	3	0	0	
117	HALFP	100	247	3	10014441765762130	0	
120	ARND	300	60	3	0	0	
121	DPROD	100	244	3	0	1	
122	EVEN	300	61	3	0	0	
123	COMPA	300	72	3	0	0	

124	LOW	300	133	3		0	0
125	NUMB	100	251	3	761014223351361524		0
126	←P1	100	246	3		0	0
127	COMPB	300	147	3		0	0
130	NUMB	100	253	3	760000000000000000		0
131	OMIT	300	225	3		0	0
132	←FOR1	300	160	3		0	0
133	←RPT1	300	222	3		0	0
134	NUMB	100	254	3	770146314631463146		0

SUBR EXTERNAL SYMBOLS REFERENCED.
REM 9021

END OF DEFINITION SET. EXTERNAL SYMBOLS.

103	V1	121	1	0	0	0
104	V2	121	2	0	0	0
105	SCNT	121	3	0	0	0
106	VPRIM	121	4	0	0	0
107	REM	110	250	0	0	0
110	INT	110	0	0	0	0
111	SUBR	10	5	0	0	0

The program NEWTN(COEF,GUESS) uses a variant of Newton's method to obtain the roots of a polynomial

$$P(X) = X^n + A_{n-1}X^{n-1} + \text{---} + A_1X + A_0$$

INPUT: Vector COEF, of length n, the coefficients $(A_{n-1}, A_{n-2}, \text{---}, A_0)$

Vector GUESS, length n, containing the approximate roots of P.

OUTPUT: COEF: unchanged

GUESS: contains the refined values of the roots

Vector POFR, length n, which contains the value of P at the next to last iteration for each root.

METHOD: Let X_K denote the value obtained for a certain root at the K-th iteration. Then

X_1 = (value obtained from GUESS)

$$X_2 = 1.001X_1$$

$$X_{K+1} = X_K - P(X_K)\,\frac{X_K - X_{K-1}}{P(X_K) - P(X_{K-1})}$$

At most twenty iterations are performed.

```
            DEFINE
            VECTORS                         COEF,GUESS,POFR
            NEWTN(COEF,GUESS). = SEQ
            INTEGERS                        J,K,L,M
            L = ROW(COEF)
            EXECUTE                         VSPACE(POFR,L)
            FOR     J = 1,1,L
            GA = GUESS_J
            FOR     K = 1,1,20
            FN = 1.0
            FOR     M = 1,1,L
            FN = COEF_M +FN×GA
            REPEAT
            CC = →INIT .I.F 1<K
            FO = FN, GO = GA
            GA = 1.001GA
            CC = →LOOP
INIT        GS = GA, DELF = FN-FO
            CC = →QUIT .I.F DELF = 0
            GA = GA-FN(GA-GO)/DELF
            GO = GS, FO = FN
LOOP        REPEAT
QUIT        GUESS_J = GA
            POFR_J = FN
            REPEAT
END
            DEFINE
LEAVE
```

```
            DEFINE
            VECTORS                         V1,V2,SCNT,VPRIME
            FUNCTIONS                       REM,INT
            REM(A,B) = A/B - INT(A/B)
            SUBR(SCLR,V1,V2,SCNT,VPRIME),=SEQ
            INTEGERS                        LNG1,LNG2,.J
            LET     HALFPI=1.570796
            LNG1=LENGTH(V1),LNG2=LENGTH(V2)
            CC=+ARND .I.F LNG1 | LNG2 .O.R LNG1 ≤ 0
            DPROD=V1 x V2
            V1 = SCLR V1
            CC=+EVEN
ARND                SLN                     +40000
EVEN        EXECUTE                         VSPACE(SCNT,LNG1)
            .J=1
COMPA       CC = +LOW .I.F REM(V1 .J,HALFPI) < 0.001
            SCNT .J = 1.0/COS(V1 .J)
            .J = .J+1
            CC = +COMPA .I.F .J≤ LNG1, +COMPB
LOW         SCNT .J = 0.0
                    SLN                     +40000
            .J = .J+1
            CC = +COMPA .I.F .J ≤ LNG1
COMPB               IF(SLF)SKP              +40000
                    TRA                     OMIT
            EXECUTE                         VSPACE(VPRIME,LNG2)
            FOR     .J = 1,1,LNG2
            VPRIME .J = V2 .J .I.F SCNT .J ± V2 .J, 0 .I.F
                                            |V2 .J| < 0.05, V1 .J
            REPEAT
OMIT                SLF                     +40000
            PRINT   SCLR,DPROD,V1,SCNT, VPRIME
END
            DEFINE
LEAVE
```

```
REM       START NEW PROGRAM.
            0    REM          1       100100002440000136
            1                 2        14000700410077763
            2                 3        121700060600000001      B
            3                 4        121700050600000000      A
            4                 5        51070007000000006
            5                 6       472164100000100001
            6                 7        140000000440100007      INT
            7                10        140007044000000000
           10                11        51070000000000006
           11                12        110400071000000004
           12                13       472164100000100001
           13                14        101000000440000137
           14                15        140006004000000000
           15                16        701000000420000000
```

```
       REM       SYMBOL TABLE.
     111         A        102      0     0                0     0
     112         B        102      1     0                0     0
     113         ←BGIN      0    113     2                0     0
```

```
       REM    EXTERNAL SYMBOLS REFERENCED.
    INT   900f
```