HEWLETT-PACKARD SOFTWARE CENTER

CONTRIBUTED PROGRAM

SNOBOL COMPILER FOR DOS/DOS-M

HP 22327E

MAY 1973

HEWLETT-PACKARD SOFTWARE CENTER
CONTRIBUTED PROGRAM DOCUMENTATION FORM

1.0 IDENTIFICATION HP 22327

1.1 Program Title

SNOBOL COMPILER FOR DOS/DOS-M

1.2 Program Language(s)

[ ] FORTRAN II [ ] FORTRAN IV

[ ] HP BASIC [ ] ALGOL

[X] Assembly Language ([X] Relocatable [ ] Absolute)

1.3 Program Type

[X] Program [ ] Subroutine [ ] Function

[ ] Other _____

2.0 PROGRAM DESCRIPTION

SNOBOL is a language translator designed for the manipulation of strings. Features of the language include symbolic naming of strings and pattern-matching. In addition to a basic set of primitive string valued functions, the system includes the facility for defining functions. These defined functions facilitate the programming of recursive procedures.

A string may be any sequence of characters. Thus, it is possible to represent a number with as many digits as we want. All arithmetic expressions can be evaluated in SNOBOL.

"EXTENDED SNOBOL3 HP FRANCE" contains some improvements: For example:
- The size of numbers is not limited and it is possible to use arithmetic expressions not completely parenthesized.
- It is possible to use not only integers but decimal numbers with unlimited precision.

(Continued next page)

CONTRIBUTED PROGRAM DOCUMENTATION FORM

2.0 PROGRAM DESCRIPTION (Continued)

SNOBOL is usable in many differents kinds of jobs. We can give as examples :

- Text formatting, text editing (see the demonstration program included).

- Text analysis (We have in Paris, a professor in one University using SNOBOL to study the manner of writing of a French author).

- High precision arithmetic (see example joined. For demonstration, we have calculated π and e with 1000 decimals).

- Automatic translation. We have one program running in Orsay :

  To translate a BASIC program into a FORTRAN IV program.

...

(continued next page)

2.0 PROGRAM DESCRIPTION (continued)

...

- Business

"EXTENDED SNOBOL3 HP FRANCE" allows to dynamically (at execution time) give the number of decimal digits to use in the representation of a number. So, it is possible very easily to write business programs even for large number manipulation.
(see one program included, written for the financial manager of HP FRANCE)

- Formal_arithmetic_and_algebra

(see included one program written in SNOBOL and allowing) :

1 - To enter a formal expression

2 - To completely parenthetize this expression

3 - To derivate the expression 2

4 - To simplify expression 3

5 - To print the formal derivate

It is constituted mainly of a compiler, a loader and an interpretor. Intermediate language is relocatable. Intermediate languages produced by BCS, DOS or DOS-M are compatible.

When DOS or DOS-M are used, it is possible with only one console (or batch) command to :

1 - Clear the job binary area

2 - Compile the program and store relocatable on jbin.

3 - Load the program

4 - Execute it

(continued next page)

CONTRIBUTED PROGRAM DOCUMENTATION FORM

2.0 (continued)

...

The program uses a sophisticated allocation of memory garbage collection technique. All output are buffered.

François Gaullier (new system analyst) and Françoise Mons (Data-Center probationer) participated to the writing of SNOBOL

END 2.0

CONTRIBUTED PROGRAM DOCUMENTATION FORM (2)

3.0 ENVIRONMENT DESCRIPTION

The environment describes the special hardware and software necessary to use the contributed program.

3.1 Supported Software Requirements - Operating System

[X] BCS [X] DOS [X] DOS-M

[ ] MTS [ ] DACE [ ] MEDACE

[ ] SIO System [ ] HP BASIC Version_____ [ ] Other_____

[ ] Educational BASIC Version_____ [ ] Self-contained

3.2 Non-Supported Software Requirements

Name and order number of subprograms called by the contributed program.

3.3 Storage Requirements

FORTRAN, ALGOL, Assembly Language (to be completed by HPSC):

(*16K)$_{10}$ without external subprograms

(*16K)$_{10}$ with external subprograms

*Only a short program (approximately 100 lines) can be written with 16K. 24K is more realistic.

BASIC: [ ] 8K [ ] 16K [ ] Other_____

3.4 Core Allocation List

[X] Enclosed [ ] N/A

3.5 Hardware Requirements

16K DOS or DOS-M System

CONTRIBUTED PROGRAM DOCUMENTATION FORM (3)

4.0 USER PROCEDURES AND DATA REQUIREMENTS

* Number of different source tapes included in this package

(programs, subprograms, test programs, data tapes, etc.)
Tapes will be strung together for distribution.
Control statement on source tape? [ ] No [X] Yes See below (Indicate)

*Source is on one mag tape in five files:

```
SNOB     ASMB,L,Z,C
SNOBØ    ASMB,L,Z,C
SNOBL    ASMB,L,Z,C
SNOBC    ASMB,L,Z
SNOBI    ASMB,L,Z
```

Assemble with X,N for Non-EAU version.

Assemble with Z for EAU version.

Usual load procedures

```
:PR,SNOB,p1,p2,p3,p4,99
```

where

p1 = Logical unit of input device (Standard is 5; set to 2 for source file input)
p2 = Logical unit of list device (Standard is 6)
p3 = Logical unit of punch device (Standard is 4)
p4 = Ø for compilation
≠ Ø for immediate RUN (p4 = logical unit of input device of SNOBOL relocatable) (Standard is Ø)
99 = The job binary parameter. If present, the object program is stored in the job binary area for immediate loading and running. Any requested punch output still occurs. (The 99 may occur anywhere in the parameter list, but terminates the list.)

p1 through p4 are optional. If not present, the standard operation is assumed.

4.0 B (continued)

EXTENDED SNOBOL3 HP FRANCE

ERROR CODES SNOBOL3

During the execution, if an error occurs, the Snobol print the following error message :

ERREUR XXXX A LA LIGNE YYY

Where

YYY is the number of line, where the error occurs

XXXX is the code number of error as fellow :

...

CODE NUMBER	SIGNIFICATION
Ø	Function failed in go to field, indicating a function call failed which evaluating a go to field
1	Memory overflow
2	Attempt to transfer to an undefined label
3	Indirect reference through the null string
4	Indirect reference through the null string, indicating an attempt to use the null string as a name
5	Indirect reference through the null string, indicating an attempt to use the null string as a name
6	Illegal operator in arithmetic operation (addition)
7	Illegal operator in arithmetic operation (soustraction)
8	Illegal operator in arithmetic operation (multiplication) 2° operator
9	Illegal operator in arithmetic operation (multiplication) 1° operator
10	Illegal operator in arithmetic operation (division)
11	Division by Ø is attempted
12	Attempt to call an undefined function
13	Function entered other than by call, indicating an attempt to return from a defined function which has not been called

14	Incorrect number of arguments during the call of a defined function
15	Too many elements in a pattern matching
16	Error in lenght specifier Ex : *F/A* where A is not integer Snobol
17	Incorrect number in the call of FLOAT (N), Ex : FLOAT("ABC") or FLOAT("1.23") or FLOAT("-125")
18	Number of decimal overflows during an arithmetic operation
19	Illegal operator in arithmetic operation (Exponentiation) Exponant is not integer
20	Illegal operator in arithmetic operation (Exponentiation) Exponant is greater than 32767
21	Illegal operator in arithmetic operation (Exponentiation) Exponant is negative with the integer mode Ex : FLOAT("Ø") X = 3.74 ↑ 1.2
22	Number of decimals overflows during an exponentiation
23	Illegal operator in arithmetic operation (Exponentiation)

4.0 E (continued)

24	Call of an defined function occurs alone on a statement
25	Attempt to output a string with more than 4096 characters

After printing this message, Snob aborts.

MESSAGES TO OPERATOR DURING COMPILATION

This message is printed on the operator console when an end-of-tape occurs on device # n.

```
I/O ERR ET EQT # n
```

EQT # n is unavailable until the operator declares it up.

```
:UP,n
:GO
```

Compilation continues after the GO.

At the end of compilation, if the 99 parameter is not present, the following message is printed :

```
$END    SNOB
```

If the 99 parameter is present and if no error occurs during compilation, the object program is loaded and it runs immediatly. If errors occurs during compilation, the following message is printed :

```
PAS D'INTERPRETATION POSSIBLE
```

and SNOB stops.

If source input for logical unit 2 (disc) is requested but no file has been declared (by a JFILE command), the system teleprinter signals :

```
$END SNOB NPRG
```

for all the other input logical units, the following messages are printed :

```
LOAD TAPE
SNOB SUSP
```

The SNOBOL suspends. The operator places a tape in the input device and types :

```
:GO
```

4.0 G (continued)

If the job binary area (where binary code is stored by a 99 parameter) overflows, compilation continues but the following message is printed on the system teleprinter :

JBIN OVF

and the object program is not loaded at the end of the compilation.

If an error is found in the SNOBOL Control statement, the following message is printed on the system teleprinter :

ØEND SNOB CS

and SNOB stops.

At the end of the execution, the SNOBOL system writtes on the system teleprinter.

SNOBOL HP FRANCE - FIN DE L'EXECUTION
ØØØXX GARBAGES

At this point, it is possible to keep the Snobol-relocatable on a relocatable file by a : STORE,R,NAME

Before the END of JOB, and after an execution with the 99 parameter, the program can be executed any time by the directive.

:PR,SNOB,,,,2,99

If the Snobol-relocatable paper tape has been punched, there is possible to execute the Snobol program by the directive, without compilation.

:PR,SNOB,,,,5

or by the directive (if the relocatable is on a relocatable file.)

:PR,SNOB,,,,2

4.0 H (continued)

SNOBOL CONTROL STATEMENT

The Control Statement must be the first statement of the source program ; it directs the compiler.

SNOBOL, p_1 , p_2

Where p_1 and p_2 are optionals and equals to B or L.

B = Binary output, snobol-relocatable is punched during the compilation. After the binary can be loaded by the command :PR,SNOB,,,,5 or :PR,SNOB,,,,2 if the binary has been stored on a relocatable file, by a :STORE,R directive.

L = List output. A listing of the source langage program is to be produced during the compilation.

CONTRIBUTED PROGRAM DOCUMENTATION FORM (4)

5.0 SUBPROGRAM INFORMATION

5.1 Entry Point(s)

N/A

CONTRIBUTED PROGRAM DOCUMENTATION FORM (5)

5.0 SUBPROGRAM INFORMATION (cont.)

5.2 Additional Exits (From a Subprogram)

N/A

6.0 SPECIAL CONSIDERATIONS

SNOBOL EXAMPLES all by PAUL GARAVINI:
HEWLETT-PACKARD FRANCE
QUARTIER DE COURTABOEUF
BOITE POSTALE 6
F - 91, ORSAY
FRANCE

7.0 EXAMPLE INPUT/OUTPUT (Test Case)

See enclosed examples.

8.0 LITERATURE REFERENCE

The Bell System Technical Journal, July-August 1966,
"The SNOBOL3 Programming Language" by D.J. Farber, R.E. Griswold,
and I.P. Polonsky (included with this documentation).
SNOBOL3 Primer, by Allen Forte, the MIT Press, Cambridge, Mass., 1967.
(Written for a person with no previous computer experience).

RELOCATING LOADER

NAME/ENTRY ADDR

```
SNOB        14117
  *SNOB     14117
  *LOGEG    14124
  *MOUVE    14141
  *RRL8     14155

SNOBO       14160
  *SNOBO    14160

SNOBL       14160
  *LECRU    14160
 .OPSY      16513
  *.OPSY    16513
 DUMRX      16553
  *SLIBR    16553
  *SLIBX    16600

SNOBC       14160
  *SNOBO    14160
 %WRIT      35417
  *%WRIT    35605
  *%WRIF    35501
  *%WBUF    35704
 .OPSY      36120
  *.OPSY    36120
 DUMRX      36160
  *SLIBR    36160
  *SLIBX    36205

SNOBI       14160
  *LONG     24332
  *DOPEN    14160
```

** THE BELL SYSTEM TECHNICAL JOURNAL, JULY-AUGUST 1966

THE SNOBOL3 PROGRAMMING LANGUAGE *

Original Manuscript By D. J. Farber, R. E. Griswold and I. P. Polonsky *

(Manuscript received March 4, 1966)

SNOBOL3 is a programming language designed for the manipulation of strings. Features of the language include symbolic naming of strings and pattern matching. In addition to a basic set of primitive string-valued functions, the system includes the facility for defining functions. These defined functions facilitate the programming of recursive procedures.

This paper presents an intuitive description of SNOBOL3 and at the same time incorporates complete reference material for the programmer.

(*)

1. INTRODUCTION

In recent years a number of high-level programming languages have been developed to extend the usefulness of the computer in dealing with primarily nonnumerical problems. The most widely used languages have been IPL, LISP, and COMIT. In 1962 SNOBOL was developed for problems involving the manipulation of character strings. The basic operations of SNOBOL permit the formation, examination, and rearrangement of strings. SNOBOL3 is a generalization and extension of SNOBOL. New features include string-valued functions and input-output facilities integrated into the string structure of the language. There are two types of functions: primitive functions that are included in the system and defined functions that are defined by the programmer in the SNOBOL3 language.

This paper is a description of SNOBOL3 as a programming language. Emphasis is placed on the language as distinct from its implementation. In order to provide information for the potential programmer, however, some references to the implementation are necessary. There are several implementations of SNOBOL3 which differ in detail, particularly with regard to input-output.

(*)

* This article has been modified for use with HP SNOBOL.

Omitted paragraphs have been denoted by an (*).

A vertical bar signifies an addition to the original article.

Areas where other implementations are likely to differ are noted in the applicable sections.

Section II describes briefly and informally the essential features of the language. This section is designed as a survey to provide an understanding of the general nature and capabilities of the language. Section III is an elaboration of Section II completing the description of the language. Sections II and III together provide a reference source for the programmer. Section IV describes the environment in which the language operates, including information which the programmer will find useful in running programs.

II INFORMAL DESCRIPTION

SNOBOL3 has just one type of basic data structure: a string of characters. The primitive operations of the language provide for the formation, examination and rearrangement of strings. Arithmetic is defined for operands that are integer or decimal strings. The operations to be performed are specified in statements that may also be labeled and may have go-to's specifying transfers. A SNOBOL3 program consists of a sequence of statements terminated by an END statement.

2.1 Names

A symbolic name can be assigned to a string and used as a means of referring to that string. There are several ways in which a name can be assigned a value. The simplest is the assignment statement. For example, the statement

```
VOWELS = 'AEIOU'
```

assigns the string AEIOU as the value of the name VOWELS. The string consisting of a pair of quotation marks enclosing a string of characters is a literal specifying the string AEIOU. The string VOWELS appearing to the left of the equal sign is a name. A name that has been assigned a value can be used whenever it is necessary to refer to that value. Thus,

```
NON.CONST = VOWELS
```

causes NON.CONST to have the same value as VOWELS. The null string, having length zero, can be assigned explicitly as value as in the statement

```
ZIP =
```

2.2 Concatenation

The basic operation of concatenation of strings is indicated by listing the names successively. The names are separated by blanks. Thus, to concatenate two string names STRING1 and STRING2 and then assign the result to the name STRING3, the following assignment statement suffices:

```
STRING3 = STRING1  STRING2
```

Many strings can be concatenated in a string expression, with literals as well as names used to specify the strings. Thus, the following rules

```
   ARGUMENT = '2X + 3'
EXPRESSION  = 'SIN('  ARGUMENT  ')'
```

would assign the string SIN(2X +3) to the name EXPRESSION.

2.3 Integer Arithmetic

Arithmetic operations can be performed on integer strings with the operators +, -, /, * having their usual meaning in integer arithmetic and ↑ indicating exponentiation. Blanks are used to separate the strings and operators. The statements

```
J = '5'
I = '3'
N = I + '2'
M = (I * '3') + J
```

assign the values 5 and 14 to the names N and M. All arithmetic operations are binary but more complex expressions can be constructed using parentheses as indicated in the last example. Arithmetic has precedence over concatenation, and both types of operations can be performed in one assignment statement. Hence, the statement

```
INDEX = 'A.' I + '1' '.' J
```

assigns the value A.4.5 to the name INDEX.

x

2.4 Pattern Matching

String pattern matching consists of examining a string for a succession of substrings of specified form. A pattern-matching statement consists of the string to be examined followed by a pattern. In its simplest form the pattern may be simply a string. For example, the statement

```
NAME.1    'IS'
```

x In HP SNOBOL arithmetic on floating point numbers is implemented, see 3.2.1.1 FLOAT(N).

would examine the value of NAME.1 to determine whether it contains the literal substring IS. The success or failure of a pattern match can affect the flow of the program and has other consequences that will be described later. In the example above, NAME.1, which specifies the string to be examined, is called the string reference of the statement. The string reference can also be a literal as in the following pattern-matching statement.

```
'+-*/'  OPERATOR
```

There are a variety of types of patterns in SNOBOL3 enabling the programmer to make complex inquiries about a string. The pattern, for example, may be expressed as a concatenation of strings as in the statement

```
EXPRESSION  'X'  OPERATOR  '1'
```

Patterns of greater generality may be obtained by using string variables. As the name indicates, a string variable may have a string as value. There are several types of string variables and the strings which are acceptable values of a string variable depend on the type of the variable. The simplest type of string variable is the arbitrary string variable, so named because it can have any string as its value. An arbitrary string variable is designated by a name bounded by asterisks.

A typical example of the use of an arbitrary string variable would be in determining whether the value of NAME.1 contains the string THE and the string IS but not necessarily consecutively. The arbitrary string variable would be used to match the substring between THE and IS. The pattern-matching statement could be

```
NAME.1  'THE'  *SEPARATOR*  'IS'
```

If the value of NAME.1 were THERAPIST, then the pattern match would be successful with *SEPARATOR* matching RAP.

A consequence of the successful pattern match is the naming of substrings that match string variables. In the above example, SEPARATOR would be given the value RAP as if the assignment statement

```
SEPARATOR = 'RAP'
```

had been executed.

NOTE: String variables are always enclosed in **

In addition to arbitrary string variables, there are two other types of string variables: fixed-length and balanced.

A fixed-length string variable can match any string of a specified number of characters. The notation for a fixed-length variable is similar to

an arbitrary string variable, except the name is followed by a slash and then by a string specifying the length.

The first three characters of the string named TEXT could be named PART1 by the rule

```
TEXT  *PART1/'3'*
```

If N had the value 3, the statement could have been written

```
TEXT  *PART1/N*
```

As a second example, consider the statement

```
'+-*'  *PLUS/'1'*  *MINUS/'1'*  *STAR/'1'*
```

The pattern successfully matches the string, the PLUS, MINUS, and STAR are assigned values.

A balanced string variable can only match strings that are parenthesis balanced in the usual algebraic sense. Strings matched by balanced string variables do not have to contain parentheses but cannot be null. Such variables are therefore useful for pattern matching on strings that are mathematical expressions. The notation for a balanced string variable consists of a name enclosed within parentheses and surrounded by a pair of asterisks. For example, if EXPRESSION has the value SIN(A*(B + C)), then the pattern match in the statement

```
EXPRESSION  'SIN('  *(ARG)*  ')'
```

is successful and ARG is given the value A*(B + C). This use of the balanced string variable may be compared to the arbitrary string variable in the following example

```
EXPRESSION 'SIN('  *ARG1*  ')'
```

where the value A*(B + C would be assigned to ARG1.

2.5 Rearranging Strings

By combining the operations of scanning and assignment in the same rule, strings may be modified by replacement, deletion, or rearrangement. In particular, if a pattern is followed by an equal sign and then by a string expression, the substring matching the pattern will be replaced by the value of the expression if the pattern match succeeds. As an example of replacement, consider the following sequence of rules.

```
CARD = 'KING OF HEARTS'
CARD   'HEART'  =  'DIAMOND'
```

The second statement causes HEART to be replaced by DIAMOND producing the string KING OF DIAMONDS. The following example illustrates how the naming of substrings by string variables may be used in the expression that specifies the rearrangement. The statements

```
SUM = 'A1+A2'
SUM   *X* '+' *Y* = '+(' X ',' Y ')'
```

change the value of SUM to +(A1,A2).

2.6 Indirect Referencing

A level of indirectness can be introduced in SNOBOL3 by prefixing a $ to a name. Thus, if DAY has the value TUESDAY, $DAY is equivalent to TUESDAY. An example of the usefulness of this facility is the ability to modify the naming done in a pattern match. Thus, in the following statements

```
DAY = 'TUESDAY'
TEXT    ',' *$DAY*  ','
```

the name TUESDAY will be assigned to a substring of TEXT if the value of TEXT is such that the pattern match succeeds.

A $ can also be prefixed to a string expression that is enclosed in parentheses. For example, the following statements assign the value of WORD to one of the names LISTA, LISTB, ... LISTZ according to the first character in the value.

```
WORD  *CH/'1'*
$('LIST' CH) = WORD
```

Thus, if WORD has the value DALLAS, the first statement sets the value of CH equal to D. The parenthesized expression

```
('LIST' CH)
```

has the value LISTD. Hence, the effect of the $ is to make the second statement equivalent to

```
LISTD = WORD
```

2.7 Labels and the Flow of Control

A label may be assigned to a statement for reference when controlling the flow of the program. The label is merely appended to the beginning of the statement as in

```
HERE LIST = '(A,B,C,D)'
```

A statement without a label must begin with a blank.

Statements in a SNOBOL3 program are executed in sequential order. This order of execution can be modified by means of a go-to appended to the end of a statement. Go-to's are separated from the rest of the statement by a slash. There are two types of go-to: unconditional and conditional. The unconditional go-to consists of a label enclosed within parentheses. Thus, after executing

```
HERE LIST = '(A,B,C,D)'    /(THERE)
```

control is transferred to the statement with label THERE. By means of the conditional go-to, control can be transferred depending on whether success or failure has been signaled during the execution of the statement. The letters S and F are used to indicate the two conditions. For example, in the following rule the transfer to the statement with label L2 will occur only if the pattern match is successful.

```
L1   TEXT   ','  *A*  '.'   /S(12)
```

If the pattern match fails, the next statement in the program is executed.

Transfer on a failure signal can be similarly programmed. As an example, consider the following sequence of statements which will delete from the string named TEXT all occurrences of the characters in LIST:

```
L1   LIST   *CHAR/'1'* = /F(DONE)
L2   TEXT    CHAR =      /S(L2)F(L1)
DONE
```

In statement L1, the first character in LIST is named CHAR and is deleted by replacing it with a null string. Statement L2 is executed repeatedly until all occurrences of CHAR have been deleted from TEXT. Then the process iterates using the next character in LIST. Finally, when there are no characters left in LIST, the pattern match in statement L1 fails. Thus, if TEXT had the value A+B*C/D+E and LIST the value +*/-, the resulting value of TEXT would be ABCDE.

A transfer may be computed by the use of indirectness in the go-to as illustrated in the following example: If PDL is assumed to have the value A1,B5,C3, then the statement

```
PDL   *RET*  ',' = /S($RET)
```

causes deletion of A1, and transfer to the statement labeled A1.

2.8 Functions

There are two types of functions in SNOBOL3: primitive and defined. Some functions may signal failure. This failure terminates the execution

of the statement in which the function call occurs and may be used to control the flow of the program.

2.8.1 Primitive Functions

A basic set of primitive functions, programmed at the machine-language level, has been included in the SNOBOL3 system.

SIZE is an example of a primitive function. The value of SIZE(X) is the number of characters in the string named X. Thus, the statements

```
STR = 'FOUL'
  Z = SIZE(STR)
```

assign the value 4 to Z. As a result of the statements

```
X = SIZE(TEXT) - '1'
TEXT  *FRONT/X*  *LAST*
```

LAST is defined to be the name of the last character in TEXT.

One use of functions is to conditionally signal failure and hence alter the flow of control. For example, EQUALS(X,Y) signals failure if X and Y do not have identical values. If the values of X and Y are identical, the function returns the null string as value. Thus, the statement

```
N = EQUALS(A,B) N + '1'
```

will increment N only if the values of A and B are equal.

Another type of primitive function is one that modifies the behavior of the SNOBOL3 system itself. The function call MODE('ANCHOR') is an example. It modifies the pattern matching processor and returns a null value. Subsequently a pattern match can succeed only if the matching substring is at the beginning of the string reference. Thus, if MODE('ANCHOR') has been executed before the statements

```
EXP = 'SIN(A + B)'
EXP '(' *(ARG)* ')'
```

the pattern match fails.

2.8.2 Defined Functions

A section of SNOBOL3 program can be defined to be a function and certain names occurring in the section can be declared formal parameters. This function declaration is accomplished by a call of the primitive function DEFINE. For example,

```
DEFINE('REVERSE(X)','REV')
```

declares the section of program beginning at the statement with label REV to be a function named REVERSE with a formal parameter X. Suppose REVERSE(X) returns as value the string named X with the characters reversed. Then the portion of program defining REVERSE could be

```
REV     X             *CHAR/'1'*   =      /F(RETURN)
        REVERSE   =     CHAR       REVERSE      /(REV)
```

The reversed label RETURN causes return to the place at which the function was called. The name of the function, in this case REVERSE, serves a special purpose. When the function is called, its value is saved and then set to the null string. When transfer to RETURN occurs, its value is the value returned by the function. Thus,

```
Z = REVERSE('ABCDE')
```

assigns the value EDCBA to Z.

2.9 Statement Format

A SNOBOL3 statement has a simple format consisting of several fields of arbitrary length separated by blanks. The fields are the label, string reference, pattern, equal sign, replacement expression, and the go-to. A statement may contain some or all of these fields. The replacement statement

```
HERE    TEXT  ' '   *WORD*   ' ' = ' '      /S(GOT)
```

has all of the fields.

If the label is omitted the statement must begin with a blank. If the next field is not a go-to, it is considered to be the string reference. Thus,

```
                              /(L7.3)
THERE                         /(HERE)
```

are statements that do not have a string reference. The statement

```
L5                    EQUALS(OP,'END')   /S(END)
```

has the string reference EQUALS(OP,'END').

The field following the string reference up to an equal sign or a go-to is the pattern. In a statement without a go-to or equal sign, the pattern is the field following the string reference. Thus, in each of the following statements

```
        STR   A  *B*  ','  =  B   /S(GREAT)
        STR   A  *B*  ','  =
   L1   STR   A  *B*  ','         /S(GREAT)
        STR   A  *B*  ','
```

the pattern is

```
        A  *B*  ','
```

Note that the elements within the pattern are also separated by blanks.

A statement without a pattern, but containing an equal sign, is an assignment statement. Some examples are:

```
        ANS = N + '5'
        RES =             /(READ)
```

The latter example assigns the null string as the value of RES.

No fields are permitted after the go-to field.

III. DETAILED DESCRIPTION

The previous section was an informal description of the basic parts of SNOBOL3. The following section completes this description in a more comprehensive and detailed manner.

3.1 Names and String Expressions

3.1.1 Names

Names are used to refer to string values symbolically. In addition, names are required for certain parts of statements:

(i) string variable names
(ii) string references in assignment statements
(iii) labels in go-to fields.

Names may be explicit or implicit. Explicit names can consist only of letters, numbers, periods and colons. Examples are:

```
N
STATEMENT.VARIABLE
X:1
37
```

Implicit names, constructed by indirect references, may consist of any nonnull string of characters. Any indirect reference is an implicit name.

For example:

```
$F
$SIZE(N)
$('M'K)
```

Consequently,

```
,,,, = '6'
```

is syntactically incorrect, but

```
 INAME = ',,,,'
$INAME = '6'
```

is proper.

The particular characters comprising a name have no significance; a name is merely an identifier. A name may be the same as a label or the name of a function.

3.1.2 String Expressions

The basic string-valued elements are:

(i) literals
(ii) names
(iii) function calls
(iv) arithmetic operations
(v) parenthetical groupings.

Any string of characters (including the null string) not containing a quotation mark (see Section 3.1.3) may be included between the quotation marks of a literal. Function calls, parenthetical groupings, and names may be indirectly referenced. Parentheses are required between successive levels of indirect references.

A string expression is a string-valued element or the concatenation of several such elements. Some typical string expressions are:

```
'PARAGRAPH SUB-HEADINGS FOLLOW'
'N'((A + '1') * INTERVAL)
$BASE + SIZE(N)
F(X,F(X,X))
$($($ROOT))
M '.' P
$('N' I)
```

3.1.3 Names and Values

All names have null values at the beginning of program execution except for the string QUOTE. QUOTE has a preassigned value which is a quotation mark.

Names, including QUOTE, may subsequently be given other values by assignment statements or as a result of pattern matching. The resulting name-value relationship between strings forms the basic data structure in SNOBOL3. Structures can be built to arbitrary depths. For example, the statements

```
N1 = 'N2'
N3 = 'N2'
N2 = 'N4'
N4 = 'N6'
N5 = 'N4'
N6 = 'N3'
```

might be used to represent relationships between data as indicated in Fig. 1.

Indirect referencing can be used to refer to the relationships in the structure. The range of such structures is limited by the fact that a name can have at most one value at any time, while a string can be the value of any number of names simultaneously.

3.2 Arithmetic

3.2.1 Integers

Some strings have the property of being SNOBOL3 integers. Such strings are required in arithmetic operations and as arguments of certain primitive functions. In order for a string to be a SNOBOL3 integer

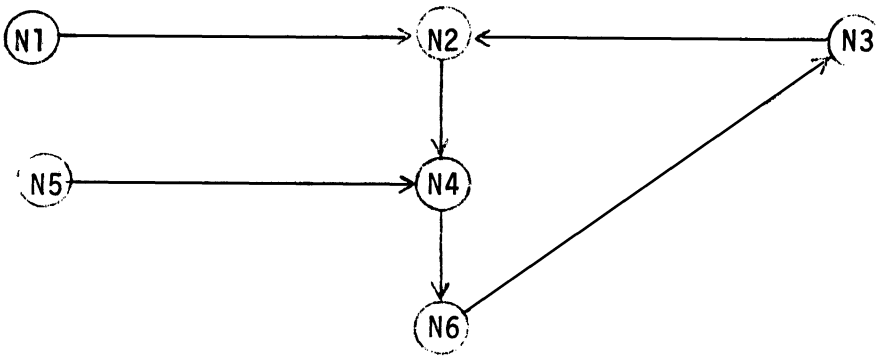


Fig. 1 - The name-value relationship among data.

(i) it must consist entirely of digits except for the first character which may be a sign, and
(ii) its absolute value considered as a decimal integer must be less than $10^{32767} - 1$

In numerical contexts
(i) unsigned numbers are taken to be positive,
(ii) leading zeros are ignored,
(iii) minus zero is equal to plus zero, and
(iv) the null string is taken to be zero.

The following strings are SNOBOL3 integers:

```
    5
   +10
0003976
   -37
-000003
   +0
```

The following strings are not SNOBOL3 integers:

```
     -
    +A
3.27E-2
   3.7
    0-                                    (*)
10,000
```

If in the integer mode (FLOAT('0') then the primitive function .NUM(X) succeeds, returning a null value, if the value of X is a SNOBOL3 integer and fails otherwise. Thus, .NUM('A') fails, while .NUM('100') returns a null value. In decimal mode .NUM succeeds if X is a SNOBOL3 decimal number.

3.2.2 Floating Point Numbers

There is a new primitive function, FLOAT(N) which permits to compute on decimal numbers. N may be any arithmetic expression but the result of this expression must be a positive integer.

Examples of correct decimal number:

```
127
127.
0035
0045.835
-57.95
.00837
```

A SNOBOL3 decimal number is constituted of any sequence of digits including or not one decimal point.

After calling the function FLOAT(N), each operation is performed with N figures after the decimal point. Ex:

```
SIZE  ('10' / '3')
```

will be = 152 if the last calling to the primitive function FLOAT was:

The integer mode may be restored by executing FLOAT ('Ø').
In this case:

'10' x '3.1415925' resulted an error.

It is possible to call at any time the function FLOAT. Before the function is called, the program is executed in the integer mode. FLOAT(N) terminates execution with an error message if N is not a positive integer less than 32767.

3.2.3 Arithmetic Expressions

Arithmetic operations must be separated from their operands by blanks. Consequently A+B is syntactically incorrect. Any expression whose value is a SNOBOL3 integer is an acceptable operand.

(*)

In extended SNOBOL3 HP FRANCE language, it is not necessary to use parentheses for grouping terms. The priorities of the operators are the same as in ALGOL or FORTRAN languages. Ex:

A x B / C - D is interpreted as (((A x B) / C) - D)

Parentheses may be used for grouping terms to create more complicated expressions but is not necessary in HP SNOBOL:

N + ('3'*'2')

In expressions containing both concatenation and arithmetic, arithmetic has precedence over concatenation. Thus, the value of

'N' '5' + '7'

is N12 and the value of

'3'*'2' '10' / '2'

is 65. Parentheses may be used to group concatenations and arithmetic to obtain the desired result. Thus, the value of

'3'*('2' '10' / '2')

is 75.

The following sequence of statements illustrates possible combinations:

```
ALPHA = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    N = SIZE(SLPHA) + '1'
    M = (N + SIZE(N)) * '2'
    K = ('-' NM) + '5'
```

As a result of executing these statement, N would have the value 27,
M the value 58, and K the value -2753.

(*)

Integers are normalized as follows:
(i) positive integers are unsigned,
(ii) leading zeros are removed, and
(iii) any value equal to zero is returned as an unsigned zero.

Thus,

```
'+0003' + '0'
```

has value 3, and

```
'' * '2'
```

has the value 0.

(*)

3.2.4 Error Conditions in Arithmetic Operations

Error conditions in arithmetic operations occur if:
(i) a fractional part would occur in integer mode,
(ii) an operand is not a SNOBOL3 integer, or decimal number
(iii) division by zero is attempted. (*)

In all cases, an error message is given, and the program is aborted.

3.2.5 Numerical Functions

There are six functions for comparing the magnitude of integers or decimal numbers:

```
.EQ(X,Y)    (X = Y)
.NE(X,Y)    (X ≠ Y)
.LT(X,Y)    (X < Y)
.LE(X,Y)    (X ≦ Y)
.GT(X,Y)    (X > Y)
.GE(X,Y)    (X ≧ Y).
```

These functions succeed, returning a null value, if the condition indicated is satisfied and fail otherwise. The functions also fail if either argument is not a SNOBOL3 integer or decimal number. A common use of the functions is to control loops. For example, the following program assigns the squares of the first 100 positive integers to the names SQ1 through SQ100, respectively.

```
        N = '1'
COMPUTE $('SQ'N) = N * N
        N = .LT(N,'100') N + '1'  /S(COMPUTE)
```

The function .REMDR(X,Y) has as its value the remainder of X divided by Y. For example, the value of

```
.REMDR('5','2')
```

is 1. The sign of the remainder is the same as the sign of the divisor and the value is normalized.

If Y is zero the program is aborted.

3.3 Pattern Matching

Pattern matching is a basic operation in SNOBOL3. The examination, rearrangement, and combination of data depend on pattern matching; and the success or failure of matching is often used for altering the flow of control.

3.3.1 Pattern Elements

A pattern consists of a succession of pattern elements separated by blanks. There are two basic categories of pattern elements: string constants and string variables.

Any string expression is a constant, except that arithmetic expressions must be enclosed in parentheses. The following expressions are examples of string constants:

```
K
'35R'
SIZE(Z)
.LE(N,M + SIZE(L))
(M + (N * '2'))
```

String variables may or may not have associated names. The following elements are examples of string variables:

```
**
*()*
*/"3"*
*VARIABLE1*
*$SIZE(N)*
*(EXP)*
```

The length of a fixed-length variable may be any string constant whose value is a nonnegative SNOBOL3 integer when evaluated. The following fixed-length variables illustrate possible forms the length may take:

```
*FL/N*
*V/(SIZE(N) + '1')*
*V/(M + (N * Z))*
```

The lengths of the following variables are syntactically incorrect:

(*)

```
*SPAN/'A'*        (The value of the length must be an integer.)
```

3.3.2 The Matching Process

Pattern matching consists of three phases:
(i) evaluation of expressions in the pattern,
(ii) the actual matching, and
(iii) the assignment of values to names associated with string variables.

3.3.2.1 Evaluation. Before any matching, all expressions in the pattern are evaluated. Expressions may occur in string constants, the names of string variables, and in the length of fixed-length variables. Evaluation proceeds from left to right. Any failure in evaluation (such as the failure of a function call or arithmetic operation) terminates the execution of the rule without any matching or naming.

The value of all expressions is fixed by evaluation before matching. No evaluation is performed during matching. The only exception to this rule is back referencing described in a following section. Thus, in the pattern

```
*N*   *SPAN/N*
```

the length of the fixed-length variable is the value of N before matching and is not influenced by any subsequent match for the arbitrary string variable with the name N.

3.3.2.2 Matching. Pattern elements must match consecutive substrings in the value of the string reference. In most cases the match can easily be determined from the following rule:

Pattern matching proceeds from left to right, each pattern element matching the shortest possible substring according to the type of the element.

In some complicated cases, more precise definitions are necessary. The following definitions provide the details for resolving difficult cases.

(i) The pattern match proceeds element by element from left to right starting at the leftmost (first) element. The elements must match consecutive substrings in the value of the string reference.

(ii) An attempt is first made to match the first element starting at the first character in the value of the string reference. If this is not possible, an attempt is made starting at the second character, and so on.

(iii) When an element is successfully matched, a forward match is attempted for the next element.

(iv) If an element cannot be matched, rematch is attempted for the preceding element. Rematching is an attempt to extend the substring matching a pattern element and occurs because the pattern match cannot be successfully concluded with the previous match.

(v) Pattern matching terminates successfully when the rightmost (last) pattern element has been matched. Pattern matching terminates in failure if no match can be found for the first element.

The methods of forward matching and rematching depend on the type of the pattern element. In each case, the element must match a substring in the string reference starting at the character following the substring matching the preceding element. The details follow.

(a) String Constants

In forward matching, a string constant matches a substring identical to its value. If this is not possible, forward matching fails. A null constant always matches.

No rematch is possible, and rematching always fails.

See the special case of back referencing.

(b) Arbitrary String Variables

In forward matching, ab arbitrary variable matches a null string.

In rematching, one character is added to the substring previously matched by the variable. If the string reference is not long enough for such a match, rematching fails.

As a special case, if the last element in the pattern is an arbitrary string variable, it matches the remainder of the string.

(c) Balanced String Variables

In forward matching, the string matched by a balanced variable depends on the first character of the substring where the variable is to match. If this first character is not a parenthesis, then the variable matches that character. If the first character is a right parenthesis, the match fails. If the first character is a left parenthesis, the string being examined is considered character by character until a matching right parenthesis is found. If there is no matching parenthesis, failure is indicated. Notice that a balanced string variable always matches at least one character.

In rematching, the previously matched substring is extended by the next shortest balanced string according to the rules for forward matching. If this is not possible, rematching fails.

(d) Fixed-Length String Variables

In forward matching, a fixed-length variable matches a substring of length specified by the variable. If the string being examined is not long enough, forward matching fails.

Rematching always fails.

(e) Back Referencing

Back referencing is a special case in pattern matching in which tentatively matched substrings can be referred to dynamically during the matching process. If a constant in the pattern has the same name as a name associated with a variable to the left of it in the pattern, the value of the constant is taken to be the substring currently matched by the variable. Thus, in the pattern

```
*N* ',' N
```

the constant N must match a substring identical to the substring matching *N*. Since matching is done left to right, a tentative match always exists for a back-referenced variable.

Back referencing only occurs when the name associated with a variable is to the left of a constant with the same name. Consequently the pattern

```
N ',' *N*
```

does not contain back referencing.

If there are several occurrences of the same name in a pattern, a

named constant back references the variable with its name which is nearest to it on the left. In the pattern

```
*N* ',' N N *N* ',' N
```

the first and second named constants refer to the first variable and the third named constant refers to the second variable.

Any type of variable may be back referenced and any number of named constants may back reference variables in an arbitrarily complicated way.

The determination of back referencing within a pattern is made after the evaluation of expressions in the pattern but before matching. In the statements

```
A = 'C'
B *C* ',' $A
```

the pattern is back referenced. However, in the statements

```
VARI = 'SPAN'
X  *VARI*  $VARI
```

there is no back referencing.

Back referencing only applies to names which are pattern elements and not to any other name in the pattern. Specifically in the pattern

```
*N*  *INT/SIZE(N)*
```

the length of INT is determined by evaluation before matching and does not change during the matching process.

3.3.2.3 Naming. If the pattern match fails, no naming is done and the execution of the rule is terminated. If the pattern match succeeds, naming is performed from left to right for each name associated with a string variable. The substring matching the variable becomes the new value of the associated name. If a name is associated with more than one variable, the value is assigned corresponding to the rightmost variable with that associated name.

In the case that a name is computed as the result of an expression, the name is determined by the evaluation made before pattern matching. Thus, in the statements

```
A = 'C'
Z  *A* ',' *$A*
```

the name associated with the second string variable is C regardless of the value of Z.

3.3.3 Pattern Matching Modes

In the normal mode of pattern matching, the first element of the pattern may match starting anywhere in the value of the string reference. Thus, the simple match

```
'0123456789'  '6'
```

succeeds. This mode is referred to as unanchored. The alternative mode, in which the first pattern element must match a substring beginning with the first character of the string reference, is called anchored. This mode may be invoked by executing the function call

```
MODE('ANCHOR')
```

Subsequently, all pattern matching will be in the anchored mode unless otherwise modified. The normal mode may be restored by

```
MODE('UNANCHOR')
```

(*)

3.3.4 Examples of Pattern Matching

The following examples illustrate some of the situations which occur in pattern matching. String reference values are given as literals for clarity. Naming is indicated for those pattern matches which succeed. The normal unanchored matching mode is assumed.

Example 1:

```
'K)AK(A + B + C)ST'    'K'    *(A)*   'ST'
```

The match succeeds with

```
A = "(A + B + C)'
```

(*)

Example 2:

```
MODE('ANCHOR') 'K)AK(A + B + C)ST'  'K'  *(A)*  'ST'
```

The match fails.

Example 3:

```
'S)(S + A*B(S'      'S'    *(A)*    'S'
```

The match fails.

Example 4:

```
'ABCDEFGHIJKLMNO'    *HV/'5'*    *A*    'K'    *B*
```

The match succeeds with

```
HV = 'ABCDE'
 A = 'FGHIJ'
 B = 'LMNO'
```

Notice that since the last pattern element is an arbitrary string variable it matches the remainder of the string reference.

Example 5:

```
'364:'    *A*    *SUM/'3'*    ':'
```

The match succeeds with

```
  A = ''
SUM = '364'
```

Example 6:

```
'ARMY'    *A*    *B*    *C*
```

The match succeeds with

```
A = ''
B = ''
C = 'ARMY'
```

Notice that the first two arbitrary string variables match null strings since this satisfies the requirement for matching the shortest possible substrings.

Example 7:

```
'ABC'   *(BAL1)*   *(BAL2)*
```

The match succeeds with

```
BAL1 = 'A'
BAL2 = 'B'
```

Example 8:

```
'AB'   *(BAL1)*  *(BAL2)*   *(BAL3)*
```

The match fails since each balanced string variable must match at least one character.

Example 9:

```
'ABCD'  *S/'2'*   *T/'3'*
```

The match fails since the string being matched is not long enough.

Example 10:

```
'ABCDEFGHFGH'   *A/'3'*   A
```

The match succeeds with

```
A = 'FGH'
```

This is a simple example of back referencing.

Example 11:

```
MODE('ANCHOR')
'ABCDEFGHFGH'   *A/'3'*  A
```

The match fails.

Example 12:

```
'32579.97'   *A*   *B*   '.'   B   A
```

The match succeeds with

```
A = '7'
B = '9'
```

These values can be verified by carefully applying the matching rules. (The expected match might be a null value for both A and B.)

Example 13:

The following example illustrates the complexity which may occur with back referencing.

```
'BACCABACABABACACAB'  *A*  *(B)*  *(C)*
                  *D*  C   D  B  D  C  A  *E*  A  E
```

The match succeeds with

```
A = ''
B = 'BAC'
C = 'CAB'
D = 'A'
E = ''
```

Example 14:

```
'A,A,B,B'   *X*  ','   X  ','  *X*  ','  X
```

The match succeeds with

```
X = 'B'
```

3.4 Program Structure and the Flow of Control

A program consists of a succession of statements terminated by an END statement containing the reserved label END.

(*)

Execution begins with the first statement of the program.

Statements are subsequently executed one after another unless control is transferred by means of a go-to.

3.4.1 Labels

Labels are distinguished by beginning in Column 1. A statement with no label must have a blank in Column 1. The first character of a label must be a letter or a digit. Subsequent characters may be anything but blanks. Labels are program constants; the particular characters in a label have no significance even if they resemble some other structure such as a name or a function call. Thus F(X) is a legitimate label but has no further meaning.

3.4.2 The Go-To Field

Go-to's are used to alter the ordinary sequential execution of statements. In general, a statement may be successfully completed, or failure may be indicated for a number of causes. The success or failure may be sensed and used by corresponding conditional go-to's to alter the order in which statements are executed.

A statement with an unconditional go-to may not have conditional go-to's. Furthermore, a statement may not have more than one unconditional, success or failure go-to. In statements with both success and failure go-to's, the go-to's may occur in either order.

The labels given in the go-to's must be names and transfer is made to the name (not its value). The label in a go-to may be computed by the use of implicit names resulting from indirect references. For example, in the statement

```
X = '3'  /($('R'X))
```

transfer is made to the statement with label R3. Function calls occurring in go-to's must not fail.

3.4.3 The Order of Execution Within a Statement

The order of execution of operations within a statement may be important to the programmer for two reasons:

(i) Failure of an operation within a statement terminates execution of the statement at that point so that subsequent operations are not performed.

(ii) Calls of defined functions may change the values of names which appear subsequently in the same statement.

Consequently, a detailed knowledge of when various parts of a statement are evaluated may be required to determine how a program will function. The overall order of execution within a statement is as follows:

(i) The string reference (if any) is evaluated.

(ii) The elements of the pattern (if any) are evaluated from left to right (see Section 3.3.2).

(iii) The pattern match (if any) is performed.

(iv) Any naming as the result of a successful pattern match is performed.

(v) If a string expression is specified as a replacement, that string expression is evaluated.

(vi) Reformation (if specified) of the value of the string reference is made.

(vii) The go-to (if any) corresponding to the success or failure of the statement is evaluated.
(viii) Transfer is made to the next statement accordingly.

If failure is signaled in any of the steps (i) through (vi) above, execution of the statement terminates at that point and the appropriate go-to is evaluated. In particular note that only the appropriate go-to is evaluated. The order of evaluation within a string expression is as follows:

(i) Elements in a concatenation are evaluated left to right.
(ii) In a function or parenthetical grouping, the innermost expression in the nesting is evaluated first.
(iii) Arithmetic is performed before concatenation.
(iv) All arguments of a function are evaluated, left to right, before the function is called.

3.4.4 Termination of Execution

Program execution is usually terminated by a transfer to the label END or by flowing into the END statement. (*)

3.5 Input-Output and File Manipulation

3.5.1 Implementation Differences

Input-output is particularly subject to differences in machines and monitor systems. Consequently the input-output behavior of the SNOBOL3 system may vary considerably in different implementations. Reference to files, record sizes and the handling of end-of-file differ most. The following sections should be read with this in mind.

3.5.2 String-Oriented Input and Output

Input and output is accomplished through string names associated with logical files.

For example, SYSPOT ("system peripheral output tape") is associated with the standard output file. Every time SYSPOT is given a value, a copy of the value is printed on the system output file. Thus, the statement

```
SYSPOT = 'TABLE OF VALUES'
```

will cause the printing of TABLE OF VALUES on the output listing.

SYSPPT ("system peripheral punch tape") is associated with the standard punch file. Values given SYSPPT are punched rather than printed.

Similarly, SYSPIT ("system peripheral input tape") is associated with the standard input file. Every time the value of SYSPIT is required, a card image is read from the input file to become the value of SYSPIT. For example, the statement

```
SYSPIT  *FIELD1*  ','  *FIELD2*  ''
```

might be used to read and name data items on input cards with the format indicated by the pattern.

SYSPLT System peripheral list output (Logical Unit : 6)

SYSPRT System peripheral reader tape (Logical Unit : 5)

which have the same meaning that SYSPIT, SYSPOT and SYSPPT which are same as:

SYSPOT System peripheral output tape (Logical Unit : 1)

SYSPPT System peripheral punch-tape (Logical Unit : 4)

SYSPIT System peripheral input tape (Logical Unit : 1)

(*)

3.5.4 Output

SYSPOT = A or SYSPLT = A or SYSPPT = A

The number of characters of the string A must be less than 4096.
All outputs use a buffering technic very sophisticated.
All output requests are stacked in a circular buffer and executed simultaneously with the execution of the program. During a garbage collection (very fast) buffering outputs are suspended.

The following functions are available only if SNOBOL3 is assembled with the N or Z option in the control statement. They significantly limit the size of a user's application program in a 16K DOS-M system.

EXIO (LU, CODE, D)

The function EXIO can have two or three parameters. The third parameter D does not have to be the true name of the string variable name when it is present.

1st parameter:	The logical unit
2nd parameter:	The code function
3rd parameter:	The name of the string variable

Code Function

For the following codes only two parameters are necessary in the call to EXIO.

CODE = 11:	Write end-of-file	(Magnetic tape)
CODE = 12:	Backspace on record	(Magnetic tape)
CODE = 13:	Forward space on record	(Magnetic tape)
CODE = 14:	Rewind	(Magnetic tape)
CODE = 15:	Rewind Standby	(Magnetic tape)
CODE = 17:	Set end-of-paper tape	
CODE = 18:	Generate paper tape leader	

The following 3 codes require the parameter D.

CODE = 1: Read:
D has the value of the input logical unit LU.

CODE = 2: Write:
D has the value of the output logical unit LU.

CODE = 3: Open file:
Value of D = name of the source S file; fails if file name not recognized.

If you try to read without opening a file, EXIO fails and D = null string. At the end of a file read, EXIO fails and D = the null string and reinitializes the open file.

MEMOIRE $(Z_1, Z_2 \ldots . Z_{512})$

Restores disc-resident strings to core memory

DISQUE $(Z_1, Z_2 \ldots Z_{512})$

Z_i = names of strings

After execution of function DISQUE all strings named in the parameter list are made disc resident.

DISQUE may be called at any time during execution and multiple calls are possible.

If Z_i is already disc-resident, no action taken, no error message.

Z_i may be defined or undefined when DISQUE is called. If Z_i is undefined, it will be put on the disc when it is defined. If Z_i is redefined the disc will be modified automatically.

3.5.5 Input

- SYSPIT or SYSPRT fails if an EOF or an end of tape are encountered.

- To read a tape, it's possible to execute the following program:

```
DEBUT       SYSPOT =  'LOAD YOUR TAPE AND TYPE : PRET'
            A = SYSPIT
            EQUALS (A, ''PRET')   /F(DEBUT)
LEC         EQUALS (TEXT)       /F (SUIT)
LEC1        TEXT = TEXT  SYSPRT  /S(LEC1)F(LEC)
```

3.6 Primitive Functions

3.6.1 Function Calls

Function calls may occur anywhere in a statement where a string value is appropriate. An argument of a function may be any string expression, however complicated. Any argument may be explicitly null and trailing arguments that are omitted are given null values. Thus, the two function calls

(*)

```
EQUALS(X,)
EQUALS(X)
```

are equivalent. (*)

All function calls return strings as value if they succeed. In the case of functions that have no natural value, a null string is returned.

It is important to notice that the function name and the left parenthesis may not be separated by blanks. Thus,

```
SIZE(N)
```

is a function call, while

```
SIZE (N)
```

is the concatenation of a name and a parenthetical grouping. Similarly, functions which have no argument must be written with the parentheses. Otherwise they will be taken for string names rather than function calls. (*)

The primitive function .NUM may be called with up to 511 arguments. (*)

3.6.3 Miscellaneous Primitive Functions

There are six primitive functions in addition to the functions described elsewhere in Section III. They are:

(i) EQUALS(X,Y) EQUALS returns a null value if the value of X is identical to the value of Y and fails otherwise. The values must be identical and not just numerically equal (compare Section 3.2.5).

(ii) UNEQL(X,Y) UNEQL returns a null value if the value of X is not identical to the value of Y and fails otherwise.

(vi) SIZE(S) SIZE returns as value the number of characters in the value of S. For example, the value of SIZE ("0123456789") is 10. (*)

(*)

3.7 Defined Functions

3.7.1 The Definition of a Function

A defined function is characterized by four items:

(i) a name, by which it is called and which is used for returning value,

(ii) a list of formal arguments, used for passing values to the function,

(iii) a label, indicating its entry point, and

(iv) a list of local names used by the function.

A function must be defined during program execution before it is used. This definition is accomplished by a call of the DEFINE function which

establishes the four items above. The form of the call is

```
DEFINE(FORM,LABEL,NAMES)
```

FORM is a prototype of the function call, giving the function name and the list of formal arguments. The value of LABEL is the entry point, and the value of NAMES is the list of local names separated by commas. For example,

```
DEFINE('FACT(N)','F')
```

defines a function FACT with one formal argument N. Execution of FACT is to begin at the label F. No local names are declared. Similarly,

```
DEFINE('MATRIXADD(A,B)','MA','I,J,K')
```

defines a function MATRIXADD with the two formal arguments A and B, the entry point MA and local names I,J, and K.

The total number of formal arguments and local names must not exceed ten. This limit is an assembly parameter.

3.7.2 The Execution of Defined Functions

The call of a defined function is identical to the call of a primitive function (see Section 3.6.1). Hence, trailing arguments which are omitted are given null values. A defined function, however, may not be called with more arguments than given in its definition.

When a defined function is called, values of the following names are saved:

(i) the name of the function.
(ii) all formal arguments.
(iii) all local names.

New values are assigned to these names as follows:

(i) the name of the function is given a null value.
(ii) the formal arguments are assigned values by evaluating the corresponding arguments in the function call.
(iii) all local names are assigned null values.

Saving of old values and assignment of new values is made from left to right as the names appear in the DEFINE call.

After these new values have been assigned, control is transferred to the entry point of the function and program execution continues in a normal fashion until transfer is made to one of the two reserved labels RETURN or FRETURN.

RETURN terminates execution of the function. By convention, the

value of the function call is the value of the function name when the return was made. For example, if FACT as defined above is designed to compute the factorial of a number, the corresponding program might be

```
F FACT = .EQ(N,'0') '1'    /S(RETURN)
  FACT = FACT(N - '1')  *N  /(RETURN)
```

Then the statements

```
SYSPOT = FACT ('3')
SYSPOT = FACT('2') + FACT('4')
```

would print 6 and 26 respectively.

FRETURN terminates execution of the function and signals failure. Execution of the statement in which the call occurs terminates at that point in the same manner as in the failure of a primitive function.

When return is made from a function (by either RETURN or FRETURN) the saved values of all names are restored in the opposite order from which they were saved.

A function may have a formal argument which is the same as its name. This is useful when the value of a function is to be a simple modification of one of its arguments. A function whose value is its first argument with all occurrences of its second argument deleted might be defined as

```
DEFINE('DELETE(DELETE,CHAR)','DEL')
```

with the program

```
DEL   DELETE   CHAR  =  /S(DEL)F(RETURN)
```

Here DELETE can be operated on as desired and the value has the correct name (that is, the name of the function) when the deletion is completed.

3.7.3 Local Names

Local names may be declared when names used in a function have values which should not be destroyed by a function call. Consider the following function which intersperses commas between the characters in its argument

```
COMMA ARG   *CHAR/'1'* = /F(RETURN)
    COMMA = COMMA CHAR ',' /(COMMA)
```

The definition would be

```
DEFINE('COMMA(ARG)','COMMA','CHAR')
```

so that the use of CHAR during the function call would not change the value of CHAR outside the function.

Local names are particularly important when recursively called functions use names for intermediate computation. (*)

IV. OPERATING ENVIRONMENT

The SNOBOL3 system consists of a compiler and an interpreter. The compiler translates SNOBOL3 source programs into an internal language suitable for the interpreter. See HP documentation form for User procedures.

4.1 Compilation

4.1.1 Source Program Listing

During compilation, the source program is read and compiled line by line. Only columns 1 through 72 are read by the compiler. Consecutive statement numbers are added to the listing for reference.

4.1.2 Comments

A line with an asterisk in column 1 is treated as a comment. Comments are printed but otherwise ignored by the compiler. Comments may be used freely throughout the program and may be placed anywhere before the END card.

4.1.3 Continuation line (cards)

A statement may be broken over line boundaries by use of the continuation line convention. A period in column 1 is interpreted by the compiler as an indication that the line is a continuation of the preceding statement. Statements may be broken over line boundaries anywhere. (*)

For example,

```
      SYSPOT = 'THE MAXIMUM LENGTH OF'
.     'THE COMPUTATIONAL THREAD HAS BEEN'
.     'COMPUTED TO BE'
```

There is no limit to the number of continue lines which may be used for a statement, but in the same instruction the number of characters must not be over 1300.

Remainder omitted. (*)

SNOBOL EXAMPLES

all by PAUL GARAVINI:

HEWLETT-PACKARD FRANCE

QUARTIER DE COURTABOEUF

BOITE POSTALE 6

F - 91, ORSAY

FRANCE

tel, (1) 920 88 01
telex, 60048

```
1  SNOBOL,L
2  *
3  ***************************************************************
4  *  THIS PROGRAM READS IN TEXT AND PRINTS N CHARACTERS         *
5  *  PER COLUMN. N AND THE NUMBER OF COLUMNS MAY BE ENTERED     *
6  *             AT EXECUTION TIME.                              *
7  ***************************************************************
8  *
9  *
10           MODE('ANCHOR')
11           DEFINE('INSERT(K,LINE)',,'IM',,'BLANK,WORD')
12           SYSPOT = 'TYPE THE VALUE OF N : _'
13           N = SYSPIT
14           SYSPOT = 'TYPE THE NUMBER OF COLUMNS : _'
15           CLMN = SYSPIT
16           .GT(((N + '3') * CLMN),'132')                  /S(ERR1)
17 *
18 *  READ IN THE TEXT
19 *
20 LEC1      SYSPOT = 'TEXTE  FILE NAME  ?  _'
21           EXIO('2',,'3',SYSPIT)          /F(LEC1)
22           SYSPLT = '1'
23           SYSPLT =
24           SYSPLT = ' ORIGINAL TEXT : '
25           SYSPLT =
26           SYSPLT =
27           NDX = '1'
28           TEXT =
29           TEXT1 =
30           FLAG = '0'
31 LEC       EXIO('2',,'1',,TXT)     /F(LEC)
32 LEC2      SYSPLT = ' ' TXT
33 POINTS    TXT *PREF* ',' *TXT*            /F(SUITE)
34           PHRSE = PHRSE ' ' PREF ','                  /(POINTS)
35 SUITE     TXT = PHRSE ' ' TXT
36           PHRSE =
37 VIRG      TXT *PREF* ',' *TXT*                /F(SUITE1)
38           PHRSE = PHRSE ' ' PREF ','          /(VIRG)
39 SUITE1    PHRSE = PHRSE ' ' TXT
40 SLASH     PHRSE *PHRSE* '/' *RELICA*                  /S(TXTCMPLT)
41           TEXT1 = TEXT1 ' ' PHRSE
42           PHRSE =
43           EXIO('2',,'1',,TXT)       /S(LEC2)F(TSTFIN)
44 *
45 *       DEBUT DE LA DEUXIEME PASS
46 *
47 TSTFIN    FLAG = '1'                          /(DCPE)
48 MPRS      SYSPLT = '1'
49           SYSPLT =
50           SYSPLT = ' EDITED TEXT:  ' N ' CHARACTERS PER COLUMN'
51           SYSPLT = '                ' CLMN ' COLUMNS. '
52           SYSPLT =
53           NDX = NDX - '1'
54           SYSPLT =                          /(DIVLOOP)
55 *
56 TXTCMPLT  TEXT1 = TEXT1 ' ' PHRSE
57 DCPE      .EQ(SIZE(TEXT1),'0')              /S(TEST)
58           TEXT1 *OP/'100'* *TEXT1*            /S(BLNCS)
59           OP = TEXT1
60           TEXT1 =
```

```
 61  BLNCS      OP *A* '  ' *R* = A ' ' B  /S(BLNCS)
 62             TEXT = TEXT OP                  /(OCPE)
 63  TEST       .GT(SIZE(TEXT),N)              /F(LASTLINE)
 64             K = '0'
 65  SCAN       TEXT *LINE/(N - K)* ' ' =      /F(BUMPK)
 66             LIGNE = ' ' INSERT(K,LINE)
 67             LIGNE *LIGNE* '/' *RESTE*      /F(TSTLGN)
 68             TEXT = '   ' RESTE ' ' TEXT
 69  TSTLGN     .EQ(SIZE(LIGNE),(N + '1'))          /S(COMPLETE)
 70             LIGNE = LIGNE ' '                  /(TSTLGN)
 71  COMPLETE   $('I' NDX) = LIGNE
 72             NDX = NDX + '1'               /(TEST)
 73  BUMPK      K = .LT(K,N) K + '1'           /S(SCAN)F(ERR)
 74  *
 75  *   FUNCTION TO INTERSPERSE K BLANKS IN A LINE
 76  *
 77  IM         INSERT = .EQ(K,'0') LINE       /S(RETURN)
 78             LINE ** ' '                    /S(BLINK)
 79             INSERT = LINE                  /(RETURN)
 80  BLINK      BLANK = BLANK ' '
 81  LOOP       LINE *WORD* BLANK =            /F(MORE)
 82             INSERT = INSERT WORD BLANK ' '
 83             K = .GT(K,'1') K - '1'         /S(LOOP)
 84             INSERT = INSERT LINE           /(RETURN)
 85  MORE       LINE = INSERT LINE
 86             INSERT =                       /(BLINK)
 87  LASTLINE   .EQ(SIZE(TEXT),N)                    /S(CMPLT)
 88             TEXT = TEXT ' '                      /(LASTLINE)
 89  CMPLT      $('I' NDX) = ' ' TEXT
 90             NDX = NDX + '1'
 91             TEXT1 =
 92             TEXT = '  '
 93             PHRSE = RELICA
 94             .EQ(FLAG,'1')                  /F(SLASH)S(MPRS)
 95  *
 96  *       BOUCLE D'IMPRESSION
 97  *
 98  DIVLOOP    NDX = .NE(.REMDR(NDX,CLMN))  NDX + '1'   /F(COLONNES)
 99             $('I' NDX) =          /(DIVLOOP)
100  COLONNES   NBLINE = NDX / CLMN
101             NDX = '1'
102  *
103  *
104  PRNTLOOP   IK = '0'
105             CHN =
106  LINLOOP    CHN = CHN '  ' $('I' (NDX + IK * NBLINE))
107             IK =  .NE(IK,CLMN)   IK + '1'          /S(LINLOOP)
108             SYSPLT = CHN
109             NDX = .NE(NDX,NBLINE)  NDX + '1'    /S(PRNTLOOP)
110             SYSPLT = '1'              /(END)
111  ERR1       SYSPOT = ' SORRY,THE LINE IS TOO BIG...' /(END)
112  ERR        SYSPOT = ' SORRY,N IS TOO SMALL '
113  END

FIN DE COMPILATION . NOMBRE D'ERREURS:      0

NOMBRE DE MOTS OCCUPES PAR LE PROGRAMME:  1458
```

ORIGINAL TEXT :

LA SOCIETE HEWLETT-PACKARD QUI REGROUPE QUINZE MILLE PERSONNES
DANS LE MONDE, CONSTRUIT DEPUIS PLUSIEURS ANNEES DES ORDINATEURS
DE PETITE ET MOYENNE PUISSANCE. TROIS MILLE DE CES MACHINES ONT
ETE VENDUES DONT PLUS D'UNE CENTAINE EN FRANCE.
LES APPLICATIONS DE CES ORDINATEURS SONT TRES VARIEES: CALCUL
SCIENTIFIQUE,GESTION,CONTROLE DE PROCESSUS INDUSTRIELS ETC.../
NOS ORDINATEURS DISPOSENT D'UN SOFTWARE TRES COMPLET.LES LANGAGES
DE PROGRAMMATION HABITUELS, ASSEMBLEUR, FORTRAN, ALGOL, BASIC SONT
DISPONIBLES. DES MONITEURS, BANDE PAPIER, BANDE MAGNETIQUE, DISQUE
ET TEMPS REEL PERMETTENT LA COMPILATION ET L'EXECUTION DES PROGRAMMES.
CEPENDANT, CES LANGAGES SONT MAL ADAPTES AUX PROBLEMES DE RECONNAISSANCE
DE CARACTERES, DE JUSTIFICATION DE TEXTE, DE RECHERCHE DE STRUCTURE.
AUSSI,UN NOUVEAU LANGAGE,SNOBOL3 A-T-IL ETE INTRODUIT POUR PERMETTRE
DE RESOUDRE AISEMENT CES PROBLEMES.//
SNOBOL3 EST CONCU POUR LA MANIPULATION DE CHAINES DE CARACTERES. IL
PERMET DE DEFINIR DE TELLES CHAINES, DE LES ASSOCIER ET AUSSI D'EFFECTUER
UNE RECHERCHE POUR SAVOIR SI UNE CHAINE DE CARACTERES EST CONTENUE
DANS UNE AUTRE. LES CHAINES PEUVENT BIEN ENTENDUES ETRE LES LIGNES
D'UN TEXTE. ON CONCOIT QU'IL SOIT ALORS FACILE DE CONSTRUIRE DANS
CE LANGAGE UN PROGRAMME FAISANT LA JUSTIFICATION D'UN TEXTE ET SA
MISE EN PAGE. LE PROGRAMME QUI A PERMIS LA JUSTIFICATION SUR 3
COLONNES DE CE TEXTE NE DEMANDE
PAS PLUS D'UNE CENTAINE D'INSTRUCTIONS!!!/
CE PROGRAMME NECESSITE SEULEMENT UN ORDINATEUR HEWLETT-PACKARD
DE 16.000 MOTS DE 16 BITS,UNE TELETYPE,UN LECTEUR ET UN PERFORATEUR
RAPIDES DE RUBAN. UNE UTILISATION PLUS SOUPLE PEUT ETRE OBTENUE
EN ADJOIGNANT UN DISQUE MAGNETIQUE./////
POUR LES SPECIALISTES://
LE NOUVEL ORDINATEUR HP 2100 QUI SERA PRESENTE POUR LA PREMIERE
FOIS EN EUROPE AU SICOB EST ENTIEREMENT COMPATIBLE AVEC LA GAMME
PRECEDENTE HP 2114,2116.PLUS RAPIDE QUE SES PREDECESSEURS,IL
EST D'UN COUT INFERIEUR GRACE A L'UTILISATION D'UNE MEMOIRE MORTE
MICROPROGRAMMEE.UNE OPTION VIRGULE FLOTTANTE CABLEE POUR LE CALCUL
SCIENTIFIQUE EST DISPONIBLE. CET ORDINATEUR NE NECESSITE
PAS D'INSTALLATION SPECIALE. COMPACT, IL PEUT MEME
ETRE POSE SUR UNE TABLE !
UNE TRES LARGE GAMME DE PERIPHERIQUES PEUT LUI ETRE
CONNECTE: DISQUES ET BANDES MAGNETIQUES,IMPRIMANTES,
TERMINAUX DE VISUALISATION,ETC...BENEFICIANT
INTEGRALEMENT DE LA PROGRAMMATION DEVELOPPEE POUR LA
SERIE PRECEDENTE,CE NOUVEL ORDINATEUR EST DONC
DES A PRESENT OPERATIONNEL. LE LANGAGE SNOBOL LUI
APPORTE DES FACILITES SUPPLEMENTAIRES POUR LE CALCUL
SCIENTIFIQUE ET L'EDITION DE TEXTES.

EDITED TEXT: 45 CHARACTERS PER COLUMN
2 COLUMNS.

LA SOCIETE HEWLETT-PACKARD QUI REGROUPE QUINZE MILLE PERSONNES DANS LE MONDE, CONSTRUIT DEPUIS PLUSIEURS ANNEES DES ORDINATEURS DE PETITE ET MOYENNE PUISSANCE. TROIS MILLE DE CES MACHINES ONT ETE VENDUES DONT PLUS D'UNE CENTAINE EN FRANCE. LES APPLICATIONS DE CES ORDINATEURS SONT TRES VARIEES: CALCUL SCIENTIFIQUE, GESTION, CONTROLE DE PROCESSUS INDUSTRIELS ETC. . .

NOS ORDINATEURS DISPOSENT D'UN SOFTWARE TRES COMPLET. LES LANGAGES DE PROGRAMMATION HABITUELS, ASSEMBLEUR, FORTRAN, ALGOL, BASIC SONT DISPONIBLES. DES MONITEURS, BANDE PAPIER, BANDE MAGNETIQUE, DISQUE ET TEMPS REEL PERMETTENT LA COMPILATION ET L'EXECUTION DES PROGRAMMES. CEPENDANT, CES LANGAGES SONT MAL ADAPTES AUX PROBLEMES DE RECONNAISSANCE DE CARACTERES, DE JUSTIFICATION DE TEXTE, DE RECHERCHE DE STRUCTURE. AUSSI, UN NOUVEAU LANGAGE, SNOBOL3 A-T-IL ETE INTRODUIT POUR PERMETTRE DE RESOUDRE AISEMENT CES PROBLEMES.

SNOBOL3 EST CONCU POUR LA MANIPULATION DE CHAINES DE CARACTERES. IL PERMET DE DEFINIR DE TELLES CHAINES, DE LES ASSOCIER ET AUSSI D'EFFECTUER UNE RECHERCHE POUR SAVOIR SI UNE CHAINE DE CARACTERES EST CONTENUE DANS UNE AUTRE. LES CHAINES PEUVENT BIEN ENTENDUES ETRE LES LIGNES D'UN TEXTE. ON CONCOIT QU'IL SOIT ALORS FACILE DE CONSTRUIRE DANS CE LANGAGE UN PROGRAMME FAISANT LA JUSTIFICATION D'UN TEXTE ET SA MISE EN PAGE. LE PROGRAMME QUI A PERMIS LA JUSTIFICATION SUR 3 COLONNES DE CE TEXTE NE DEMANDE PAS PLUS D'UNE CENTAINE D'INSTRUCTIONS!!!

CE PROGRAMME NECESSITE SEULEMENT UN ORDINATEUR HEWLETT-PACKARD DE 16. 000 MOTS DE 16 BITS, UNE TELETYPE, UN LECTEUR ET UN PERFORATEUR RAPIDES DE RUBAN. UNE UTILISATION PLUS SOUPLE PEUT ETRE OBTENUE EN ADJOIGNANT UN DISQUE MAGNETIQUE.

POUR LES SPECIALISTES:

LE NOUVEL ORDINATEUR HP 2100 QUI SERA PRESENTE POUR LA PREMIERE FOIS EN EUROPE AU SICOB EST ENTIEREMENT COMPATIBLE AVEC LA GAMME PRECEDENTE HP 2114, 2116. PLUS RAPIDE QUE SES PREDECESSEURS, IL EST D'UN COUT INFERIEUR GRACE A L'UTILISATION D'UNE MEMOIRE MORTE MICROPROGRAMMEE. UNE OPTION VIRGULE FLOTTANTE CABLEE POUR LE CALCUL SCIENTIFIQUE EST DISPONIBLE. CET ORDINATEUR NE NECESSITE PAS D'INSTALLATION SPECIALE. COMPACT, IL PEUT MEME ETRE POSE SUR UNE TABLE ! UNE TRES LARGE GAMME DE PERIPHERIQUES PEUT LUI ETRE CONNECTE: DISQUES ET BANDES MAGNETIQUES, IMPRIMANTES, TERMINAUX DE VISUALISATION, ETC. . . BENEFICIANT INTEGRALEMENT DE LA PROGRAMMATION DEVELOPPEE POUR LA SERIE PRECEDENTE, CE NOUVEL ORDINATEUR EST DONC DES A PRESENT OPERATIONNEL. LE LANGAGE SNOBOL LUI APPORTE DES FACILITES SUPPLEMENTAIRES POUR LE CALCUL SCIENTIFIQUE ET L'EDITION DE TEXTES.

EDITED TEXT: 95 CHARACTERS PER COLUMN
1 COLUMNS.

LA SOCIETE HEWLETT-PACKARD QUI REGROUPE QUINZE MILLE PERSONNES DANS LE MONDE, CONSTRUIT DEPUIS PLUSIEURS ANNEES DES ORDINATEURS DE PETITE ET MOYENNE PUISSANCE. TROIS MILLE DE CES MACHINES ONT ETE VENDUES DONT PLUS D'UNE CENTAINE EN FRANCE. LES APPLICATIONS DE CES ORDINATEURS SONT TRES VARIEES: CALCUL SCIENTIFIQUE, GESTION, CONTROLE DE PROCESSUS INDUSTRIELS ETC. . .

NOS ORDINATEURS DISPOSENT D'UN SOFTWARE TRES COMPLET. LES LANGAGES DE PROGRAMMATION HABITUELS, ASSEMBLEUR, FORTRAN, ALGOL, BASIC SONT DISPONIBLES. DES MONITEURS, BANDE PAPIER, BANDE MAGNETIQUE, DISQUE ET TEMPS REEL PERMETTENT LA COMPILATION ET L'EXECUTION DES PROGRAMMES. CEPENDANT, CES LANGAGES SONT MAL ADAPTES AUX PROBLEMES DE RECONNAISSANCE DE CARACTERES, DE JUSTIFICATION DE TEXTE, DE RECHERCHE DE STRUCTURE. AUSSI, UN NOUVEAU LANGAGE, SNOBOL3 A-T-IL ETE INTRODUIT POUR PERMETTRE DE RESOUDRE AISEMENT CES PROBLEMES.

SNOBOL3 EST CONCU POUR LA MANIPULATION DE CHAINES DE CARACTERES. IL PERMET DE DEFINIR DE TELLES CHAINES, DE LES ASSOCIER ET AUSSI D'EFFECTUER UNE RECHERCHE POUR SAVOIR SI UNE CHAINE DE CARACTERES EST CONTENUE DANS UNE AUTRE. LES CHAINES PEUVENT BIEN ENTENDUES ETRE LES LIGNES D'UN TEXTE. ON CONCOIT QU'IL SOIT ALORS FACILE DE CONSTRUIRE DANS CE LANGAGE UN PROGRAMME FAISANT LA JUSTIFICATION D'UN TEXTE ET SA MISE EN PAGE. LE PROGRAMME QUI A PERMIS LA JUSTIFICATION SUR 3 COLONNES DE CE TEXTE NE DEMANDE PAS PLUS D'UNE CENTAINE D'INSTRUCTIONS!!!

CE PROGRAMME NECESSITE SEULEMENT UN ORDINATEUR HEWLETT-PACKARD DE 16. 000 MOTS DE 16 BITS, UNE TELETYPE, UN LECTEUR ET UN PERFORATEUR RAPIDES DE RUBAN. UNE UTILISATION PLUS SOUPLE PEUT ETRE OBTENUE EN ADJOIGNANT UN DISQUE MAGNETIQUE.

POUR LES SPECIALISTES:

LE NOUVEL ORDINATEUR HP 2100 QUI SERA PRESENTE POUR LA PREMIERE FOIS EN EUROPE AU SICOB EST ENTIEREMENT COMPATIBLE AVEC LA GAMME PRECEDENTE HP 2114, 2116. PLUS RAPIDE QUE SES PREDECESSEURS, IL EST D'UN COUT INFERIEUR GRACE A L'UTILISATION D'UNE MEMOIRE MORTE MICROPROGRAMMEE. UNE OPTION VIRGULE FLOTTANTE CABLEE POUR LE CALCUL SCIENTIFIQUE EST DISPONIBLE. CET ORDINATEUR NE NECESSITE PAS D'INSTALLATION SPECIALE. COMPACT, IL PEUT MEME ETRE POSE SUR UNE TABLE ! UNE TRES LARGE GAMME DE PERIPHERIQUES PEUT LUI ETRE CONNECTE: DISQUES ET BANDES MAGNETIQUES, IMPRIMANTES, TERMINAUX DE VISUALISATION, ETC. . . BENEFICIANT INTEGRALEMENT DE LA PROGRAMMATION DEVELOPPEE POUR LA SERIE PRECEDENTE, CE NOUVEL ORDINATEUR EST DONC DES A PRESENT OPERATIONNEL. LE LANGAGE SNOBOL LUI APPORTE DES FACILITES SUPPLEMENTAIRES POUR LE CALCUL SCIENTIFIQUE ET L'EDITION DE TEXTES.

EDITED TEXT: 32 CHARACTERS PER COLUMN
3 COLUMNS.

LA SOCIETE HEWLETT-PACKARD QUI REGROUPE QUINZE MILLE PERSONNES DANS LE MONDE, CONSTRUIT DEPUIS PLUSIEURS ANNEES DES ORDINATEURS DE PETITE ET MOYENNE PUISSANCE. TROIS MILLE DE CES MACHINES ONT ETE VENDUES DONT PLUS D'UNE CENTAINE EN FRANCE. LES APPLICATIONS DE CES ORDINATEURS SONT TRES VARIEES: CALCUL SCIENTIFIQUE, GESTION, CONTROLE DE PROCESSUS INDUSTRIELS ETC. . .

NOS ORDINATEURS DISPOSENT D'UN SOFTWARE TRES COMPLET. LES LANGAGES DE PROGRAMMATION HABITUELS, ASSEMBLEUR, FORTRAN, ALGOL, BASIC SONT DISPONIBLES. DES MONITEURS, BANDE PAPIER, BANDE MAGNETIQUE, DISQUE ET TEMPS REEL PERMETTENT LA COMPILATION ET L'EXECUTION DES PROGRAMMES. CEPENDANT, CES LANGAGES SONT MAL ADAPTES AUX PROBLEMES DE RECONNAISSANCE DE CARACTERES, DE JUSTIFICATION DE TEXTE, DE RECHERCHE DE STRUCTURE. AUSSI, UN NOUVEAU LANGAGE, SNOBOL3 A-T-IL ETE INTRODUIT POUR PERMETTRE DE RESOUDRE AISEMENT CES PROBLEMES.

SNOBOL3 EST CONCU POUR LA MANIPULATION DE CHAINES DE CARACTERES. IL PERMET DE DEFINIR DE TELLES CHAINES, DE LES ASSOCIER ET AUSSI D'EFFECTUER UNE RECHERCHE POUR SAVOIR SI UNE CHAINE DE CARACTERES EST CONTENUE DANS UNE AUTRE. LES CHAINES PEUVENT BIEN ENTENDUES ETRE LES LIGNES D'UN TEXTE. ON CONCOIT QU'IL SOIT ALORS FACILE DE CONSTRUIRE DANS CE LANGAGE UN PROGRAMME FAISANT LA JUSTIFICATION D'UN TEXTE ET SA MISE EN PAGE. LE PROGRAMME QUI A PERMIS LA JUSTIFICATION SUR 3 COLONNES DE CE TEXTE NE DEMANDE PAS PLUS D'UNE CENTAINE D'INSTRUCTIONS!!!

CE PROGRAMME NECESSITE SEULEMENT UN ORDINATEUR HEWLETT-PACKARD DE 16. 000 MOTS DE 16 BITS, UNE TELETYPE, UN LECTEUR ET UN PERFORATEUR RAPIDES DE RUBAN. UNE UTILISATION PLUS SOUPLE PEUT ETRE OBTENUE EN ADJOIGNANT UN DISQUE MAGNETIQUE.

POUR LES SPECIALISTES:

LE NOUVEL ORDINATEUR HP 2100 QUI SERA PRESENTE POUR LA PREMIERE FOIS EN EUROPE AU SICOB EST ENTIEREMENT COMPATIBLE AVEC LA GAMME PRECEDENTE HP 2114, 2116. PLUS RAPIDE QUE SES PREDECESSEURS, IL EST D'UN COUT INFERIEUR GRACE A L'UTILISATION D'UNE MEMOIRE MORTE MICROPROGRAMMEE. UNE OPTION VIRGULE FLOTTANTE CABLEE POUR LE CALCUL SCIENTIFIQUE EST DISPONIBLE. CET ORDINATEUR NE NECESSITE PAS D'INSTALLATION SPECIALE. COMPACT, IL PEUT MEME ETRE POSE SUR UNE TABLE ! UNE TRES LARGE GAMME DE PERIPHERIQUES PEUT LUI ETRE CONNECTE: DISQUES ET BANDES MAGNETIQUES, IMPRIMANTES, TERMINAUX DE VISUALISATION, ETC. . . BENEFICIANT INTEGRALEMENT DE LA PROGRAMMATION DEVELOPPEE POUR LA SERIE PRECEDENTE, CE NOUVEL ORDINATEUR EST DONC DES A PRESENT OPERATIONNEL. LE LANGAGE SNOBOL LUI APPORTE DES FACILITES SUPPLEMENTAIRES POUR LE CALCUL SCIENTIFIQUE ET L'EDITION DE TEXTES.

EDITED TEXT: 30 CHARACTERS PER COLUMN
4 COLUMNS.

LA SOCIETE HEWLETT-PACKARD QUI REGROUPE QUINZE MILLE PERSONNES DANS LE MONDE, CONSTRUIT DEPUIS PLUSIEURS ANNEES DES ORDINATEURS DE PETITE ET MOYENNE PUISSANCE. TROIS MILLE DE CES MACHINES ONT ETE VENDUES DONT PLUS D'UNE CENTAINE EN FRANCE. LES APPLICATIONS DE CES ORDINATEURS SONT TRES VARIEES: CALCUL SCIENTIFIQUE, GESTION, CONTROLE DE PROCESSUS INDUSTRIELS ETC. . .

NOS ORDINATEURS DISPOSENT D'UN SOFTWARE TRES COMPLET. LES LANGAGES DE PROGRAMMATION HABITUELS, ASSEMBLEUR, FORTRAN, ALGOL, BASIC SONT DISPONIBLES. DES MONITEURS, BANDE PAPIER, BANDE MAGNETIQUE, DISQUE ET TEMPS REEL PERMETTENT LA COMPILATION ET L'EXECUTION DES PROGRAMMES. CEPENDANT, CES LANGAGES SONT MAL ADAPTES AUX PROBLEMES DE RECONNAISSANCE DE CARACTERES, DE JUSTIFICATION DE TEXTE, DE RECHERCHE DE STRUCTURE. AUSSI, UN NOUVEAU LANGAGE, SNOBOL3 A-T-IL ETE INTRODUIT POUR PERMETTRE DE RESOUDRE AISEMENT CES PROBLEMES.

SNOBOL3 EST CONCU POUR LA MANIPULATION DE CHAINES DE CARACTERES. IL PERMET DE DEFINIR DE TELLES CHAINES, DE LES ASSOCIER ET AUSSI D'EFFECTUER UNE RECHERCHE POUR SAVOIR SI UNE CHAINE DE CARACTERES EST CONTENUE DANS UNE AUTRE. LES CHAINES PEUVENT BIEN ENTENDUES ETRE LES LIGNES D'UN TEXTE. ON CONCOIT QU'IL SOIT ALORS FACILE DE CONSTRUIRE DANS CE LANGAGE UN PROGRAMME FAISANT LA JUSTIFICATION D'UN TEXTE ET SA MISE EN PAGE. LE PROGRAMME QUI A PERMIS LA JUSTIFICATION SUR 3 COLONNES DE CE TEXTE NE DEMANDE PAS PLUS D'UNE CENTAINE D'INSTRUCTIONS!!!

CE PROGRAMME NECESSITE SEULEMENT UN ORDINATEUR HEWLETT-PACKARD DE 16. 000 MOTS DE 16 BITS, UNE TELETYPE, UN LECTEUR ET UN PERFORATEUR RAPIDES DE RUBAN. UNE UTILISATION PLUS SOUPLE PEUT ETRE OBTENUE EN ADJOIGNANT UN DISQUE MAGNETIQUE.

POUR LES SPECIALISTES:

LE NOUVEL ORDINATEUR HP 2100 QUI SERA PRESENTE POUR LA PREMIERE FOIS EN EUROPE AU SICOB EST ENTIEREMENT COMPATIBLE AVEC LA GAMME PRECEDENTE HP 2114, 2116. PLUS RAPIDE QUE SES PREDECESSEURS, IL EST D'UN COUT INFERIEUR GRACE A L'UTILISATION D'UNE MEMOIRE MORTE MICROPROGRAMMEE. UNE OPTION VIRGULE FLOTTANTE CABLEE POUR LE CALCUL SCIENTIFIQUE EST DISPONIBLE. CET ORDINATEUR NE NECESSITE PAS D'INSTALLATION SPECIALE. COMPACT, IL PEUT MEME ETRE POSE SUR UNE TABLE ! UNE TRES LARGE GAMME DE PERIPHERIQUES PEUT LUI ETRE CONNECTE: DISQUES ET BANDES MAGNETIQUES, IMPRIMANTES, TERMINAUX DE VISUALISATION, ETC. . . BENEFICIANT INTEGRALEMENT DE LA PROGRAMMATION DEVELOPPEE POUR LA SERIE PRECEDENTE, CE NOUVEL ORDINATEUR EST DONC DES A PRESENT OPERATIONNEL. LE LANGAGE SNOBOL LUI APPORTE DES FACILITES SUPPLEMENTAIRES POUR LE CALCUL SCIENTIFIQUE ET L'EDITION DE TEXTES.

```
  1  SNOBOL,L
  2             ESPACE = '                                                  '
  3             2 = '2'
  4  DEBUT      SYSPOT = 'VALEUR MAXI. DE N ?'
  5             SYSPLT = '1'
  6             FIN = SYSPIT
  7  DEB        N = .NE(N,FIN) N + '1'          /F(END)
  8             X = FLOAT(2 * N)  2 ^ N
  9             Y = 2 ^ ('-' N)
 10             ESPACE   *ESN/('4' - SIZE(N))*
 11             ESPACE   *ESX/('20' - SIZE(X))*
 12             SYSPLT = ' ' ESX X ' ' ESN N '   ' Y           /(DEB)
 13  END

FIN DE COMPILATION . NOMBRE D'ERREURS:     0

NOMBRE DE MOTS OCCUPES PAR LE PROGRAMME:   293
```

2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.0625
32	5	0.03125
64	6	0.015625
128	7	0.0078125
256	8	0.00390625
512	9	0.001953125
1024	10	0.0009765625
2048	11	0.00048828125
4096	12	0.000244140625
8192	13	0.0001220703125
16384	14	0.00006103515625
32768	15	0.000030517578125
65536	16	0.0000152587890625
131072	17	0.00000762939453125
262144	18	0.000003814697265625
524288	19	0.0000019073486328125
1048576	20	0.00000095367431640625
2097152	21	0.00000047683715820312 5
4194304	22	0.00000023841857910156 25
8388608	23	0.00000011920928955078125
16777216	24	0.00000005960464477539062 5
33554432	25	0.00000002980232238769531 25
67108864	26	0.00000001490116119384765625
134217728	27	0.00000000745058059692382812 5
268435456	28	0.00000000372529029846191406 25
536870912	29	0.00000000186264514923095703125
1073741824	30	0.00000000093132257461547851562 5
2147483648	31	0.00000000046566128730773925781 25
4294967296	32	0.00000000023283064365386962890625
8589934592	33	0.00000000011641532182693481445312 5
17179869184	34	0.00000000005820766091346740722656 25
34359738368	35	0.00000000002910383045673370361328125
68719476736	36	0.00000000001455191522836685180664062 5
137438953472	37	0.00000000000727595761418342590332031 25
274877906944	38	0.00000000000363797880709171295166015625
549755813888	39	0.00000000000181898940354585647583007812 5
1099511627776	40	0.00000000000090949470177292823791503906 25
2199023255552	41	0.00000000000045474735088646411895751953125
4398046511104	42	0.00000000000022737367544323205947875976562 5
8796093022208	43	0.00000000000011368683772161602973937988281 25
17592186044416	44	0.00000000000005684341886080801486968994140625
35184372088832	45	0.00000000000002842170943040400743484497070312 5
70368744177664	46	0.00000000000001421085471520200371742248535156 25
140737488355328	47	0.00000000000000710542735760100185871124267578125
281474976710656	48	0.00000000000000355271367880050092935562133789062 5
562949953421312	49	0.00000000000000177635683940025046467781066894531 25
1125899906842624	50	0.00000000000000088817841970012523233905334472656 25
2251799813685248	51	0.00000000000000044408920985006261616945266723632812 5
4503599627370496	52	0.00000000000000022204460492503130808472633618164062 5
9007199254740992	53	0.00000000000000011102230246251565404236316680908203125
18014398509481984	54	0.00000000000000005551115123125782702118158340454101562 5
36028797018963968	55	0.00000000000000002775557561562891351059079170227050781 25
72057594037927936	56	0.00000000000000001387778780781445675529539585113525390625
144115188075855872	57	0.00000000000000000693889390390722837764769792556762695312 5
288230376151711744	58	0.00000000000000000346944695195361418882384896278381347656 25
576460752303423488	59	0.00000000000000000173472347597680709441192448139190673828125

1152921504606846976	60	0.000000000000000000867361737988403547205962240695953369140625
2305843009213693952	61	0.0000000000000000004336808689942017736029811203479766845703125
4611686018427387904	62	0.0000000000000000002168404344971008868014905601739883422851562 5
9223372036854775808	63	0.0000000000000000001084202172485504434007452800869941711425781 25
18446744073709551616	64	0.0000000000000000000542101086242752217003726400434970855712890625

```
  1  SNOBOL,L
  2  DEBUT SYSPOT =
  3        SYSPOT = '      PROGRAMME DE TABULATION DE FACTORIELLE N'
  4  *
  5  *       POUR FAIRE UN SAUT DE PAGE
  6  *
  7        SYSPLT =  '1'
  8        SYSPLT =  '0'
  9        SYSPLT =  '     PROGRAMME SNOBOL DE TABULATION DE FACTORIELLE
 10  .N'
 11        SYSPOT = 'VALEUR MAXIMUM DE N  ?  _'
 12        E = SYSPIT
 13        SYSPOT = .LT(E,'0')  'JE VEUX UN NOMBRE POSITIF'  /S(DEBUT)
 14        SYSPOT = 'IL FAUT IMPRIMER A PARTIR DE  _'
 15        XIMP =  SYSPIT
 16        SYSPOT = 'COMBIEN VOULEZ-VOUS DE CARACTERES PAR LIGNE  _'
 17        XCAR = SYSPIT
 18        P = '0'
 19        X = '1'
 20  RETOUR P = .NE(P,E)  P + '1'          /F(END)
 21        X = X * P
 22        Z = X
 23        Z  =  .LT(P,XIMP)     /S(RETOUR)
 24        Z   *B/(XCAR - '30')*    =    /S(WRITE)
 25        B = Z
 26  WRITE SYSPLT =
 27        SYSPLT =
 28        SYSPLT =
 29        SYSPLT = '        FACTORIELLE (' P ')   = '  B
 30        B = .EQ(B,Z)        /S(LONG)
 31  IMP   Z =  ' '  Z
 32  IMP1  Z   *SYSPLT/XCAR*    =  ' '    /S(IMP1)
 33        SYSPLT =  Z
 34  LONG  SYSPLT =
 35        SYSPLT = '     SOIT ' SIZE(X) ' CARACTERES'    /(RETOUR)
 36  END

FIN DE COMPILATION . NOMBRE D'ERREURS:     0

NOMBRE DE MOTS OCCUPES PAR LE PROGRAMME:   544
```

PROGRAMME SNOBOL DE TABULATION DE FACTORIELLE N

FACTORIELLE (498) = 4890327959884208692189333809310809323661620612437646662093842037618618730075679349120226242493800728
909874214337267349034722960809932573949822786393424165472857344977712747910913670777900883483665522574849349604586395471636355789
423097404049910397670856871606231571742939857185495140858813762835131853592731374262506767493368887927817327619484053414766844672
206598436782375875809864052859213264653372169637005960038332251017681999898876040432572861207542204690441581193460564006510582742
775697084590120023447414503292590175178202677922346128483741177607249924846680928112732567753434414856087566940983410396816614184
129602398702861502626505259814625516813822944549402592127703705963627992787412303378681451959807425239242265402097522071882090214
638608500051382859329165963520410628104233297453470859543551613744071948846612879118916041538480733008384844598933046785629604639
562438299720075091914905365451384368266070950506884190161954563270477653027976468690636861249056062002640551120268241945243186595
102720000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000080000000000

SOIT 1129 CARACTERES

FACTORIELLE (499) = 2440273651982220137402477570846093852507148685606385684384827176771690746307763995210992895004406563
726027232954296407168326757444156354400961570410318658570955815143878661207545921718172540858349095764849825452688611340346541538
922125604620905288437757578931509554299726988735562075288548067654730794942772955756990876979191075075980846482122542653968655491
431092619954405562029122162376747419062032712648865974059127793257823317949539144175853857742563560140530349015536821439248780788
645072845210469891700259837143002497413923136283250718113386847626017712498493783128253551308963773013187695903550721788011490477
880671596952727889810626124647498132890097649330151893471724149275850368400918739385962044527943905194381890435646663513869163017
104665641525640046805253815796684903424012415429281958912232255258291902474459826680339104727701885771184037454867590346029172715
141656711560317470865537777360240799764769404302935210890815327071968348860960257876627793763278974939317635009013852730676350110
956257280000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

SOIT 1132 CARACTERES

FACTORIELLE (500) = 1220136825991110068701238785423046926253574342803192842192413588385845373153881997605496447502203281
863013616477148203584163378722078177200480785205159329285477907571939330603772960859086270429174547882424912726344305670173270769
461062802310452644218878789465754777149863494367781037644274033827365397471386477878495438489595537537990423241061271326984327745
715546309977202781014561081188373709531016356324432987029563896628911658974769572087926928871281780070265174507768410719624390394
322536422605234945850129918571501248706961568141625359056693423813008856249246891564126775654481886506593847951775360894005745238
940335798476363944905313062323749066445048824665075946735862074637925184200459369692981022263971952597190945217823331756934581508
552332820762820023402626907898342451712006207714640979456116127629145951237229913340169552363850942885592018727433795173014586357
570828355780158735432768888680120399882384702151467605445407663535984174430480128938313896881639487469658817504506926365338175055
478128640000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000

SOIT 1135 CARACTERES

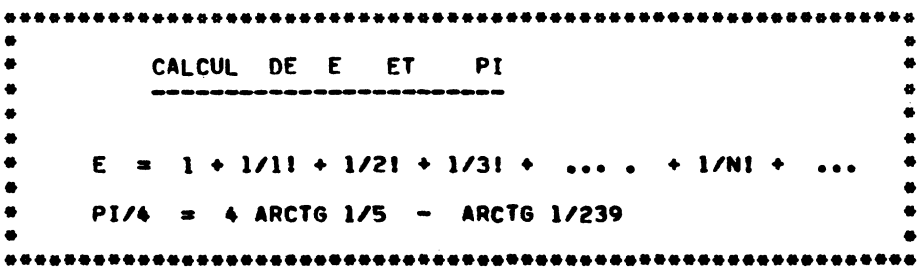
```
SNOBOL.L
*****************************************************************
*                                                               *
*      CALCUL DE E: BASE  DES  LOGARITHMES  NEPERIENS           *
*         PAR LA SOMME DE LA SERIE                              *
*                                                               *
*    E = 1 + 1/1! + 1/2! + 1/3! + 1/4! +  .....   .  + 1/N! +  ... *
*                                                               *
*****************************************************************
*
*
          DEFINE('IMPRI(Z,N,CHAIN)',,'IMPRI',,'B,NB')
          DEFINE('SAULIG(N)',,'SAUL',,'I')
          SYSPOT = 'COMBIEN DE CARACTERES PAR LIGNE VOULEZ-VOUS ? _'
          NBCAR = SYSPIT
          SYSPLT = '1'
          ESPACE = '                                                  '
          SPACE = ' '
          E = NBCAR / '2' - '33'
          ESPACE     .GT(E,'0')   *SPACE/E*          /F(SUIT)
SUIT      SYSPLT =
          SYSPOT = 'COMBIEN DE DECIMALES VOULEZ-VOUS ?    _'
          F = SYSPIT
          SYSPLT = SPACE
.'*****************************************************************'
          SYSPLT = SPACE
.'*                                                               *'
          SYSPLT = SPACE
.'*         CALCUL  DE  E   ET    PI                              *'
          SYSPLT = SPACE
.'*         ----------------------                                *'
          SYSPLT = SPACE
.'*                                                               *'
          SYSPLT = SPACE
.'*                                                               *'
          SYSPLT = SPACE
.'*     E  =  1 + 1/1! + 1/2! + 1/3! +  ... .  + 1/N! +  ...     *'
          SYSPLT = SPACE
.'*                                                               *'
          SYSPLT = SPACE
.'*    PI/4  =  4 ARCTG 1/5  -  ARCTG 1/239                       *'
          SYSPLT = SPACE
.'*                                                               *'
          SYSPLT = SPACE
.'*****************************************************************'
          SYSPOT = SAULIG('10')
*
*
          X = F + '2'
          E = '2.5'
          A = '2'
          Z1 = A
          C = FLOAT(X + '3')  '3'     /(QN1)
QN        C = .NE(Z1,E1) C + '1'         /F(FIN)
QN1       A = A * C
          Z = E
          E = E + '1' / A
          Z   *Z1/X*       /F(ERR)
          E   *E1/X*      /S(QN)
ERR       SYSPOT = .NE(C,'3') 'ERREUR  ! !'    /S(END)F(QN)
```

```
*
*
*
FIN       NB = IMPRI(Z1,NBCAR,'         E  =  ') - '2'          /(LONG)
LONG      SYSPOT = SAULIG('2')
          SYSPLT = '     SOIT ' NB ' DECIMALES EXACTES'
          SYSPOT = SAULIG('10')
*
          Z1 = FLOAT('0')
*
*
*
*****************************************************************
*                                                               *
*      CALCUL DE PI:                                            *
*        PAR LA SOMME DE LA SERIE                               *
*                                                               *
*          PI/4 =  4 ARCTG 1/5  -  ARCTG 1/239                  *
*                                                               *
*****************************************************************
*
*
*
          C = '10' ^ (F + '4')
          K = '4' * C
          SIG = '1'
          S = '0'
          U = '5'
          V = '239'
          N = '1'
A     Y = V * N
          .LE(Y,C)          /F(B)
          S = S + SIG * (K / (U * N) - C / Y)
          U = U * '25'
          V = V * '57121'
          N = N + '2'
          SIG = SIG * '-1'  /(A)
B     Y = U * N
          .LE(Y,K)          /F(FINCALPI)
          S = S + SIG * K / Y
          U = U * '25'
          N = N + '2'
          SIG = SIG * '-1'   /(B)
*
*
FINCALPI  S = '4' * S
          S */'1'* *V/F* *H* = '3.' V
          NB = IMPRI(S,NBCAR,'       PI  = ')  - '2'
          SYSPOT = SAULIG('2')
          SYSPLT = '     SOIT ' NB ' DECIMALES EXACTES'
          SYSPLT = '1'                      /(END)
*
*
*
*                 PROGRAMME  IMPRI
*
*
*
IMPRI     IMPRI = SIZE(Z)
          Z    *B/(N - SIZE(CHAIN))*    =   ' '      /S(WRITE)
```

```
121             B = Z
122  WRITE      SYSPLT =
123             SYSPLT = CHAIN B
124             B = EQUALS(B,Z)       /S(RETURN)
125  IMP        Z     *SYSPLT/N*    = ' '          /S(IMP)
126             SYSPLT = Z
127             SYSPLT =                           /(RETURN)
128  *
129  *
130  SAUL       SYSPLT = .NE(I,N)            /F(RETURN)
131             I = I + '1'             /(SAUL)
132  *
133  *
134  END
```

FIN DE COMPILATION . NOMBRE D'ERREURS: 0

NOMBRE DE MOTS OCCUPES PAR LE PROGRAMME: 1375



E = 2.718281828459045235360287471352662497757247093699959574966967627724076630353547594571382178525166427427466391932003059921817413596629043572900334295260

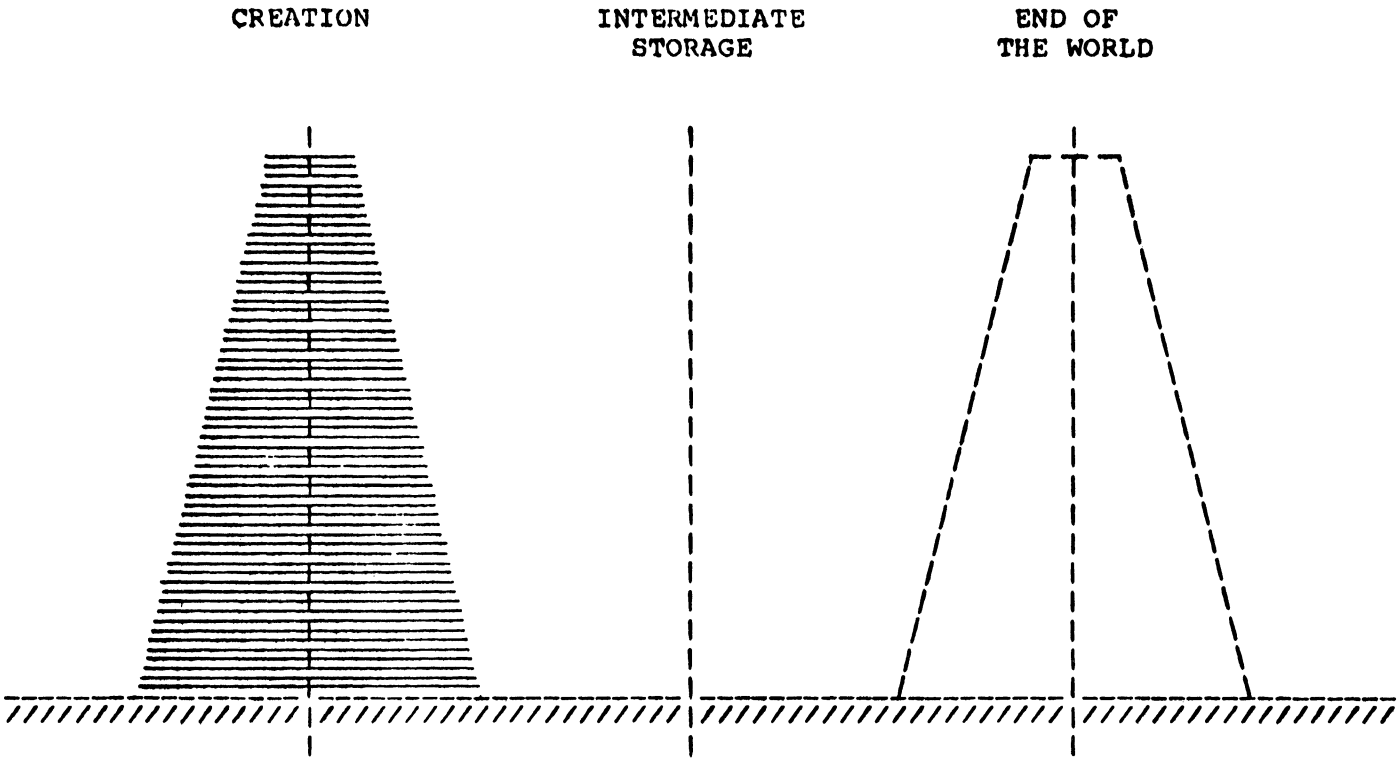
SOIT 150 DECIMALES EXACTES

PI = 3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128

SOIT 150 DECIMALES EXACTES

Example - Tower of Hanoi

The Tower of Hanoi is a game derived from the ancient Tower of Brahma, a ritual allegedly practiced by Brahman priests to predict the end of the world. At the time of creation, 64 golden discs of decreasing size appeared stacked on a diamond needle. Nearby were two other diamond needles, both empty. The Brahman priests, created at the same time, were set to the task of moving the discs from their original needle to a second needle using, when necessary, the third needle as temporary storage. Before all 64 discs are moved to the second needle and stacked in decreasing size, the end of the world will be upon us.



Movement of the discs is governed by the rules:

1) only one disc may be moved at a time,
2) a disc may be moved from any needle to any other, and
3) at no time may a larger disc rest upon a smaller disc.

A solution to the Tower of Hanoi is a recursive function which prints out the steps necessary to move N discs from one needle to another (where N is hopefully a good deal smaller than 64). A program that defines the function HANOI and tests it by moving 5 discs from needle A to needle C follows.

```
        DEFINE('HANOI(N,NS,ND,NI)')                    :(HANOI.END)
*
HANOI   EQ(N,0)                                        :S(RETURN)
        HANOI(N - 1,NS,NI,ND)
        OUTPUT  =  'MOVE DISC  ' N '  FROM  ' NS ' TO ' ND
        HANOI(N - 1,NI,ND,NS)                          :(RETURN)
HANOI.END
*
TEST    HANOI(5,'A','C','B')
END
```

```
MOVE DISC  1  FROM  A TO C
MOVE DISC  2  FROM  A TO B
MOVE DISC  1  FROM  C TO B
MOVE DISC  3  FROM  A TO C
MOVE DISC  1  FROM  B TO A
MOVE DISC  2  FROM  B TO C
MOVE DISC  1  FROM  A TO C
MOVE DISC  4  FROM  A TO B
MOVE DISC  1  FROM  C TO B
MOVE DISC  2  FROM  C TO A
MOVE DISC  1  FROM  B TO A
MOVE DISC  3  FROM  C TO B
MOVE DISC  1  FROM  A TO C
MOVE DISC  2  FROM  A TO B
MOVE DISC  1  FROM  C TO B
MOVE DISC  5  FROM  A TO C
MOVE DISC  1  FROM  B TO A
MOVE DISC  2  FROM  B TO C
MOVE DISC  1  FROM  A TO C
MOVE DISC  3  FROM  B TO A
MOVE DISC  1  FROM  C TO B
MOVE DISC  2  FROM  C TO A
MOVE DISC  1  FROM  B TO A
MOVE DISC  4  FROM  B TO C
MOVE DISC  1  FROM  A TO C
MOVE DISC  2  FROM  A TO B
MOVE DISC  1  FROM  C TO B
MOVE DISC  3  FROM  A TO C
MOVE DISC  1  FROM  B TO A
MOVE DISC  2  FROM  B TO C
MOVE DISC  1  FROM  A TO C
```

The program logic can be seen by induction. Clearly, moving no discs requires no steps. Moving one disc from needle A to needle C requires one step.

```
MOVE DISC  1  FROM  A TO C
```

Moving two discs from A to C requires three steps.

```
MOVE DISC  1  FROM  A TO B
MOVE DISC  2  FROM  A TO C
MOVE DISC  1  FROM  B TO C
```

Moving three discs from A to C requires seven steps.

```
MOVE DISC  1  FROM  A TO C
MOVE DISC  2  FROM  A TO B
MOVE DISC  1  FROM  C TO B
MOVE DISC  3  FROM  A TO C
MOVE DISC  1  FROM  B TO A
MOVE DISC  2  FROM  B TO C
MOVE DISC  1  FROM  A TO C
```

The general solution is:

```
MOVE N-1 DISCS FROM  A TO B
MOVE DISC  N  FROM  A TO C
MOVE N-1 DISCS  FROM  B TO C
```

The implementation is simple. HANOI is defined with four arguments:

1) N is the number of discs to be moved,
2) NS is the starting needle,
3) ND is the destination needle, and
4) NI is the intermediate storage needle.

On entry to HANOI, the value of N is compared with zero. If N is zero, no discs are moved and the function returns. If N is not zero, HANOI is called recursively to move N-1 discs from the starting needle to the intermediate storage needle. Having done that, the command to move the Nth disc from the starting needle to the destination needle is printed. Finally, HANOI is called a second time to move the N-1 discs from intermediate storage to the destination needle.

The SNOBOL4 programming language
by R.E. Griswold, J.F. Poage, I.P. Polonsky
Bell Telephone Laboratories, Incorporated
Prentice-Hall, Inc., Englewood Cliffs, New Jersey

```
  1  SNOBOL,L
  2  ************************************************************************
  3  *                                                                      *
  4  *        TOWER OF HANOI                                                *
  5  *        --------------                                                *
  6  *                                                                      *
  7  *   THE TOWER OF HANOI IS A GAME DERIVED FROM THE ANCIENT THOWER       *
  8  *   OF BRAHMA, A RITUAL ALLEGEDLY PRACTICED BY BRAHMAN PRIESTS         *
  9  *   TO PREDICT THE END OF WORLD. AT THE TIME OF CREATION, 64           *
 10  *   GOLDEN DISCS OF DECREASING SIZE APPEARED STACKED ON A              *
 11  *   DIAMOND NEEDLE. NEARBY WERE TWO OTHER DIAMOND NEEDLES, BOTH        *
 12  *   EMPTY. THE BRAHMAN PRIESTS, CREATED AT THE SAME TIME, WERE         *
 13  *   SET TO THE TASK OF MOVING THE DISCS FROM THEIR ORIGINAL            *
 14  *   NEEDLE TO A SECOND NEEDLE USING, WHEN NECESSARY, THE THIRD         *
 15  *   NEEDLE AS TEMPORARY STORAGE. BEFORE ALL 64 DISCS ARE MOVED         *
 16  *   TO THE SECOND NEEDLE AND STACKED IN DECREASING SIZE, THE END       *
 17  *   OF THE WORLD WILL BE UPON US.                                      *
 18  *                                                                      *
 19  *   MOVEMENT OF DISCS IS GOVERNED BY THE RULES:                        *
 20  *     (1)  ONLY ONE DISC MAY BE MOVED AT A TIME,                       *
 21  *     (2)  A DISC MAY BE MOVED FROM ANY NEEDLE TO ANY OTHER, AND       *
 22  *     (3)  AT NO TIME MAY A LARGER DISC REST UPON A SMALLER DISC       *
 23  *                                                                      *
 24  *                                                                      *
 25  *     A SOLUTION TO THE TOWER OF HANOI IS A RECURSIVE FUNCTION         *
 26  *   THAT PRINTS OUT THE STEPS NECESSARY TO MOVE  N  DISCS FROM         *
 27  *   ONE NEEDLE TO ANOTHER.                                             *
 28  *                                                                      *
 29  *                                                                      *
 30  ************************************************************************
 31  *
 32  *
 33            DEFINE('HANOI(N,NS,ND,NI)','HAN','P')
 34            SYSPLT = '1'
 35  DEBUT     SYSPOT = 'COMBIEN DE DISQUES VOULEZ VOUS ? _'
 36            K = SYSPIT
 37            SYSPOT = .LT(K,'0')  'DONC VOUS N' QUOTE 'EN VOULEZ'
 38  .    ' PLUS.'    /S(END)
 39            SYSPLT = '   JEUX  AVEC  ' K '  DISQUES'
 40            SYSPLT =
 41            SYSPLT =
 42            Z = HANOI(K,'A','C','B')
 43            SYSPLT =
 44            SYSPLT =
 45            SYSPLT =
 46            SYSPLT =                      /(DEBUT)
 47  HAN       P = .EQ(N,'0')                  /S(RETURN)
 48            Q = HANOI(N - '1',NS,NI,ND)
 49            SYSPLT = '     DISQUE ' N '   DE ' NS ' A ' ND
 50            Q = HANOI(N - '1',NI,ND,NS)  /(RETURN)
 51  END

FIN DE COMPILATION . NOMBRE D'ERREURS:     0

NOMBRE DE MOTS OCCUPES PAR LE PROGRAMME:   383
```

JEUX AVEC 3 DISQUES

```
DISQUE 1   DE A A C
DISQUE 2   DE A A B
DISQUE 1   DE C A B
DISQUE 3   DE A A C
DISQUE 1   DE B A A
DISQUE 2   DE B A C
DISQUE 1   DE A A C
```

JEUX AVEC 5 DISQUES

DISQUE 1 DE A A C
DISQUE 2 DE A A B
DISQUE 1 DE C A B
DISQUE 3 DE A A C
DISQUE 1 DE B A A
DISQUE 2 DE B A C
DISQUE 1 DE A A C
DISQUE 4 DE A A B
DISQUE 1 DE C A B
DISQUE 2 DE C A A
DISQUE 1 DE B A A
DISQUE 3 DE C A B
DISQUE 1 DE A A C
DISQUE 2 DE A A B
DISQUE 1 DE C A B
DISQUE 5 DE A A C
DISQUE 1 DE B A A
DISQUE 2 DE B A C
DISQUE 1 DE A A C
DISQUE 3 DE B A A
DISQUE 1 DE C A B
DISQUE 2 DE C A A
DISQUE 1 DE B A A
DISQUE 4 DE B A C
DISQUE 1 DE A A C
DISQUE 2 DE A A B
DISQUE 1 DE C A B
DISQUE 3 DE A A C
DISQUE 1 DE B A A
DISQUE 2 DE B A C
DISQUE 1 DE A A C

JEUX AVEC 6 DISQUES

DISQUE 1 DE A A B
DISQUE 2 DE A A C
DISQUE 1 DE B A C
DISQUE 3 DE A A B
DISQUE 1 DE C A A
DISQUE 2 DE C A B
DISQUE 1 DE A A B
DISQUE 4 DE A A C
DISQUE 1 DE B A C
DISQUE 2 DE B A A
DISQUE 1 DE C A A
DISQUE 3 DE B A C
DISQUE 1 DE A A B
DISQUE 2 DE A A C
DISQUE 1 DE B A C
DISQUE 5 DE A A B
DISQUE 1 DE C A A
DISQUE 2 DE C A B
DISQUE 1 DE A A B
DISQUE 3 DE C A A
DISQUE 1 DE B A C
DISQUE 2 DE B A A
DISQUE 1 DE C A A
DISQUE 4 DE C A B
DISQUE 1 DE A A B
DISQUE 2 DE A A C
DISQUE 1 DE B A C
DISQUE 3 DE A A B
DISQUE 1 DE C A A
DISQUE 2 DE C A B
DISQUE 1 DE A A B
DISQUE 6 DE A A C
DISQUE 1 DE B A C
DISQUE 2 DE B A A
DISQUE 1 DE C A A
DISQUE 3 DE B A C
DISQUE 1 DE A A B
DISQUE 2 DE A A C
DISQUE 1 DE B A C
DISQUE 4 DE B A A
DISQUE 1 DE C A A
DISQUE 2 DE C A B
DISQUE 1 DE A A B
DISQUE 3 DE C A A
DISQUE 1 DE B A C
DISQUE 2 DE B A A
DISQUE 1 DE C A A
DISQUE 5 DE B A C
DISQUE 1 DE A A B
DISQUE 2 DE A A C
DISQUE 1 DE B A C
DISQUE 3 DE A A B
DISQUE 1 DE C A A
DISQUE 2 DE C A B
DISQUE 1 DE A A B
DISQUE 4 DE A A C
DISQUE 1 DE B A C

```
DISQUE 2   DE B A A
DISQUE 1   DE C A A
DISQUE 3   DE B A C
DISQUE 1   DE A A B
DISQUE 2   DE A A C
DISQUE 1   DE B A C
```

JEUX AVEC 0 DISQUES