SPITBOL 360

USER'S GUIDE AND NEWSLETTERS

Spitbol 360 Version 2.3

8 November 2001

http://www.snobol4.com

About This Guide

This guide is the companion documentation to the General Public License (GPL) distribution of SPITBOL 360.

What This Document Contains

This document contains:

- Scanned page images of the original SPITBOL 360 Manual.
- Scanned page images of the original SPITBOL 360 Newsletters.

Note that this document is searchable by Acrobat® Reader™.

Related Information

For general information about SNOBOL4+, the Macro SPITBOL series of compilers, SPITBOL 360, or SPITBOL 370 please visit

http://www.snobol4.com

For technical information about SPITBOL 360 please contact

goldberg@snobol4.com

Acknowledgements

Thanks to Robert Dewar and Ken Belcher for allowing me to re-release SPITBOL 360 under the GPL for all to use and study.

Thanks to Claire Goldberg for helping her dad scan the pages.

Bob Goldberg

CONTENTS

SPITBOL

Version 2.0

Robert B. K. Dewar

Illinois Institute of Technology

February 12, 1971

S4D23

Note: This document is a reproduction of a document of the same title prepared at the Illinois Institute of Technology. It is reproduced in its present form by permission of the author to make it more widely available to individuals interested in SNOBOL4.

R. E. Griswold
Bell Telephone Laboratories, Incorporated
Holmdel, New Jersey 07733

February 16, 1971

TABLE OF CONTENTS

1. Introduction

SPITBOL is a new implementation of the SNOBOL-4 computer language for use on the IBM 360 and 370 series computers. SPITBOL is considerably smaller than the existing implementation (from Bell Telephone Laboratories, implemented by the designers of the SNOBOL-4 language -- R. E. Griswold and I. Polonsky) and has execution speeds up to ten times faster. For certain programs, notably those with in-line patterns, the gain in speed may be even greater.

Unlike BTL SNOBOL-4, SPITBOL is a true compiler which generates executable machine code. The generated code may be listed in assembly form. Of course, the complexity of the SNOBOL-4 language dictates that system subroutines be used for many common functions. SPITBOL can be run as an 'in-core' system like WATFIV, where jobs are executed as sson as they are compiled, and jobs may be bathed together. Alternately, the compiler can generate an object module for later execution.

This manual is the preliminary documentation and user's guide for SPITBOL. It is assumed that the reader is familiar with the standard version (referred to as BTL SNOBOL-4 in the remainder of the manual). Version 3.4 of SNOBOL-4 is the reference version for comparison. There are several minor incompatabilities and some features are unimplemented at the current time. In addition, there are several additions to the language in this implementation.

In general an attempt has been made to retain upward compatability wherever possible. Most SNOBOL-4 programs which operate correctly using BTL SNOBOL-4 should operate correctly when compiled and executed using SPITBOL.

SPITBOL was designed and implemented by --

Robert B. K. Dewar
and
Kenneth Belcher

Information Science Center
Illinois Institute of Technology
Chicago, Illinois 60616

2. Summary of Differences

This section contains a summary of the significant differences between SPITBOL and BTL SNOBOL-4.

2.1. Features Not Implemented

At the current time, the following features of BTL SNOBOL-4 are not implemented. It is intended that all these features will be included in later releases of the SPITBOL system.

1) Program defined trace functions

2) OPSYN for operators (third argument)

3) The BLOCK datatype (as implemented in SNOBOL-4B)

2.2. Features Implemented Differently

The following features are implemented in SPITBOL, but the usage is different from that in BTL SNOBOL-4 and changes in existing programs may be required.

1) Recovery from execution errors (see SPITBOL function SETEXIT)

2) I/O is somewhat different. The FORTRAN I/O routines are not used. However, a FORTRAN format processing routine has been included for compatibility.

3) The JCL required for running under OS/360 is different.

2.3. Additional Features

The following additional features (not in BTL SNOBOL-4) are included in the SPITBOL system.

1) The datatype DREAL (double precision real).

2) The additional functions BREAKX, LEQ, LGE, LLE, LLT, LNE, LPAD, REVERSE, RPAD, SETEXIT, SUBSTR.

3) Additional flexibility in I/O. Support of all record formats recognized by QSAM. Format free variable record length I/O allowing simple input output of strings. Support for partioned datasets and multi-file tape volumes.

4) The symbolic dump optionally includes elements of arrays, tables and program defined datatypes.

5) Both the pattern matching stack and the function call push down stack may expand to use all available dynamic memory if necessary.

2.4. Other incompatabilities

1) The value of a modifiable keyword can be changed only by direct assignment using =, pattern assignment cannot be used to change a keyword value and the name operator cannot be applied to a keyword.

2) SPITBOL allows some datatype conversions not allowed in BTL SNOBOL-4. For example, a REAL value may be used in pattern alternation and is converted to a string. In general, SPITBOL will convert objects to an appropriate datatype if at all possible.

3) The unary . (name) operator applied to a natural variable yields a NAME rather than a STRING. Since this NAME can be converted to a STRING when required, the difference is normally not noticed. The only points at which the difference will be apparent is in use of the IDENT, DIFFER and DATATYPE functions and when used as a TABLE subscript.

4) SPITBOL normally operates in an optimized mode which generates a number of incompatabilities. This mode can be turned off if necessary, see description of the control cards -OPTIMIZE and -NOOPTIMIZE.

5) SPITBOL permits leading and trailing blanks on numeric strings which are to be converted to STRING.

6) Several of the built-in functions are different. These are identified by a * on their name in section 6 of this manual.

7) SPITBOL does not permit exponentiation of two real numbers.

8) The BACKSPACE function is not implemented.

9) Deferred expressions in pattern matching are not assumed to match one character (see section 5).

3. Datatypes and Conversion

3.1. Datatypes in SPITBOL

1) STRING — Strings range in length from 0 (null string) to 32758 characters (subject to the setting of &MAXLNGTH). Any characters from the EBCDIC set can appear.

2) INTEGER — Integers are stored in 32 bit form allowing a range of -2**31 to +2**31-1. There is no negative zero.

3) REAL — Stored as a 32 bit short form floating point number.

4) DREAL — Stored using long form floating point. The low order byte is not available and is stored as zero, thus giving a 48 bit mantissa (15 decimal digits).

5) ARRAY — Arrays may have up to 255 dimensions.

6) TABLE — A table may have any number of elements, see description of the TABLE function for further details. Any SPITBOL object may be used as the name of a table element, including the null string.

7) PATTERN — Pattern structures may range up to 32768 bytes which means there is essentially no limit on the complexity of a pattern.

8) NAME — A name can be obtained from any variable. Note that in SPITBOL, the name operator (unary dot) applied to a natural variable yields a name, not a string as in BTL SNOBOL-4.

9) EXPRESSION — Any expression may be deferred.

10) CODE — A string representing a valid program can be converted to code at execution time. The resulting object, of type CODE, may be executed in the same manner as the original program.

3.2. Datatype Conversion

As far as possible, SPITBOL converts from one datatype to another as required. The following table shows which conversions are possible. A blank entry indicates that the conversion is never possible, X indicates that the conversion is always possible, and F indicates that conversion may be possible, depending on the value involved.

convert from	convert to: S	I	R	D	A	T	P	N	E	C
S	X	F	F	F			X	X	F	F
I	X	X	X	X			X	X	X	
R	X	F	X	X			X	X	X	
D	X	F	X	X			X	X	X	
A					X	X				
T					F	X				
P							X			
N	X	F	F	F					F	F
E									X	
C										X

S STRING
I INTEGER
R REAL
D DREAL
A ARRAY
T TABLE
P PATTERN
N NAME
E EXPRESSION
C CODE

The following section gives detailed descriptions for each of the possible conversions.

STRING --> INTEGER

Leading and trailing blanks are ignored. A leading sign is optional. The sign, if present, must immediately precede the digits. A null string or all blank string is converted to zero.

STRING --> REAL

Leading and trailing blanks are ignored. A leading sign, if present, must immediately precede the number. The number itself may be written in standard (FORTRAN type) format with an optional exponent. The conversion is always accurate, the last bit is correctly rounded.

STRING --> DREAL

The rules are the same as for STRING to REAL. Note that a STRING is considered to represent a DREAL if more than eight significant digits are given, or if a D is used for the exponent instead of an E. The conversion is always accurate, the last bit is correctly rounded.

STRING --> PATTERN

A pattern is created which will match the string value.

STRING --> NAME

The result is the name of the natural variable with a name of the given string. This is identical to the result of applying the unary dot operator to the variable in question. The null string cannot be converted to a name.

STRING --> EXPRESSION

The string must represent a legal SPITBOL expression. The compiler is used to convert the string into its equivalent expression and the result can be used anywhere an expression is permitted.

STRING --> CODE

The string must represent a legal SPITBOL program, complete with labels, and using semicolons to separate statements. The compiler is used to convert the string into executable code. The resulting code can be executed by transferring to it with a direct GOTO or by a normal transfer to a label within the code.

INTEGER --> STRING

The result has no leading or trailing blanks. Leading zeros are suppressed. A preceding minus sign is supplied for negative values. Zero is converted to '0'.

INTEGER --> REAL

A real number is obtained by adding a zero fractional part. Note that significance is lost in converting integers whose absolute value exceeds 2**24-1.

INTEGER --> DREAL

A DREAL is obtained by adding a zero fractional part. Significance is never lost in this conversion.

INTEGER --> PATTERN

First convert to STRING and then treat as STRING to PATTERN.

INTEGER --> NAME

First convert to STRING and then treat as STRING to NAME.

INTEGER --> EXPRESSION

The result is a expression which when evaluated yields the INTEGER as its value.

REAL --> STRING

The real number is converted to its standard character representation. Fixed type format is used if possible, otherwise an exponent (using E) is supplied. Seven significant digits are generated, the last being correctly rounded for all cases. Trailing insignificant zeros are suppressed after rounding has taken place.

REAL --> INTEGER

This conversion is only possible if the REAL is in the range permitted for integers. In this case, the result is obtained by truncating the fractional part.

REAL --> DREAL

Additional low order zeros are added to extend the mantissa.

REAL --> PATTERN

First convert to STRING and then treat as STRING to PATTERN.

REAL --> NAME

First convert to STRING and then treat as STRING to NAME.

REAL --> EXPRESSION

The result is an expression which when evaluated yields the REAL as its value.

DREAL --> STRING

Like REAL to STRING except that 15 significant digits are given and a D is used for the exponent if one is required.

DREAL --> INTEGER

This conversion is only possible if the DREAL is in the range permitted for integers. In this case, the result is obtained by truncating the fractional part.

DREAL --> REAL

The low order digits of the mantissa are truncated to reduce the precision.

DREAL --> PATTERN

First convert to STRING and then treat as STRING to PATTERN.

DREAL --> NAME

First convert to STRING and then treat as STRING to NAME.

DREAL --> EXPRESSION

The result is an expression which when evaluated yields the DREAL as its value.

ARRAY --> TABLE

The array must be two dimensional with a second dimension of two or an errors occurs. For each entry (value of the first subscript), a table entry using the (X,1) entry as name and the (X,2) entry as value is created. The table built has the same number of hash headers (see TABLE function) as the first dimension.

TABLE --> ARRAY

The table must have at least one element which is non-null. The array generated is two dimensional. The first dimension is equal to the number of non-null entries in the table. The second dimension is two. For each entry, the (X,1) element in the array is the name and the (X,2) element is the value. The order of the elements in the array is the order in which elements were put in the table.

NAME --> STRING

A NAME can be converted to a STRING only if it is the name of a natural variable. The resulting string is the character name of the variable.

NAME --> INTEGER, REAL, DREAL, PATTERN, EXPRESSION, CODE

The NAME is first converted to a string (if possible) and then the conversion proceeds as described for STRING.

4. Syntax

This section describes differences between the syntax in SPITBOL and BTL SNOBOL-4. These differences are minor and should not affect existing programs.

1) Reference to elements of arrays which are themselves elements of arrays is possible without using the item function. Thus the following are equivalent --

   A<J><K> = B<J><K>

   ITEM(A<J>,K) = ITEM(B<J>,K)

2) The full 80 columns of input may optionally be used -- see description of the -IN80 control card.

3) The only way to change the value of a keyword is by direct assignment. It is not permissible to use a keyword in any other context requiring a name.

4) The compiler permits real constants to be followed by a FORTRAN style exponent E+xxx or D+xxx, the latter signifies a double precision real (DREAL).

5. Pattern Matching

Pattern matching is essentially compatible, however there are some minor differences and extensions as described in this section.

The stack used for pattern matching can expand to fill all available dynamic memory if neccessary. Thus the diagnostic issued for an infinite pattern recursion is simply the standard memory overflow message.

In quickscan mode, deferred expressions are not assumed to match one character. This is a definite incompatability and some left recursive patterns may go berserk. However, experience seems to indicate that this heuristic has caused more problems than it has solved, so it has been abandoned.

In SPITBOL the values of &QUICKSCAN and &ANCHOR are obtained only at the start of the match. In BTL SNOBOL-4, changing these values during a match can lead to unexpected results.

The BREAKX function allows construction of an extended break pattern. See the description in the function section.

6. Functions

This section defines the functions built-in to the SPITBOL system. The functions are described in alphabetical order. In most cases, the arguments are preconverted to some particular datatype. This is indicated in the function header by the notation --

FUNCTION(STRING,INTEGER etc...

If the corresponding argument cannot be converted to the indicated datatype, an error with major code 1 (illegal datatype) occurs -- see section on error codes. In some cases, the range of arguments permitted is restricted. Arguments outside the permitted domain cause the generation of an error with major code 13 (incorrect value for function or operator). The usage 'ARGUMENT' implies that the argument can be of any datatype. 'NUMERIC' implies that any numeric datatype can occur (INTEGER, REAL or DREAL).

In the following descriptions, a single asterisk following the name of the function indicates that the implementation of the function differs from that in BTL SNOBOL-4, or that the function is not available in BTL SNOBOL-4.

6.1 ANY -- Pattern to Match Selected Char

ANY(STRING) or ANY(EXPRESSION)

This function returns a pattern which will match a single character selected from the characters in the argument string. A null argument is not permitted.

If an expression argument is used, then the expression is evaluated during the pattern match and must give a non-null string result.

6.2. APPLY* -- Apply Function

APPLY(NAME,ARG,ARG,...)

The first argument is the name of a function to be applied to the (possibly null) list of arguments following. Unlike BTL SNOBOL-4, SPITBOL does not require the number of arguments to match. Extra arguments are ignored, and missing arguments are supplied as null strings.

6.3. ARBNO -- Pattern for Iterated Match

ARBNO(PATTERN)

This function returns a pattern which will match an arbitrary number of occurrences of the pattern argument, including the null string (corresponding to zero occurrences).

6.4. ARG -- Obtain Argument Name

ARG(NAME,INTEGER)

The first argument represents the name of a function. The integer is the number of a formal argument to this function. The returned result is the string name of the selected argument. ARG fails if the integer is out of range (less than one, or greater than the number of arguments).

6.5. ARRAY -- Generate Array Structure

ARRAY(STRING,ARG)

The string represents the prototype of an array to be allocated. This is in the format 'LBD1:HBD1,LBD2:HBD2,..' The low bound (LBD) may be omitted for some or all of the dimensions, in which case a low bound of one is assumed. The second argument (of any datatype) is the initial value of all the elements in the array. If the second argument is omitted, the initial value of all elements will be the null string.

6.6. BREAK -- Construct Scanning Pattern

BREAK(STRING) or BREAK(EXPRESSION)

This function returns a pattern which will match any string up to but not including a character in the string argument. A null argument is not permitted.

If an expression argument is given, the resulting pattern causes the string to be evaluated during pattern matching. In this case, the evaluated result must be a non-null string.

6.7. BREAKX* -- Construct Scanning Pattern

BREAKX(STRING) or BREAKX(EXPRESSION)

BREAKX returns a pattern whose initial match is the same as a corresponding BREAK pattern. However, BREAKX has implicit alternatives which are obtained by scanning past the first break character found and scanning to the next break character.

Note that BREAKX may be used to replace ARB in many situations where BREAK cannot be used easily. For example the following replacement can be made --

ARB ('CAT' | 'DOG') ---> BREAKX('CD') ('CAT' | 'DOG')

In the case of an expression argument, the expression is evaluated during pattern matching and must yield a non-null string value. Note that the evaluation of the expression is not repeated on rematch attempts by extension.

6.8. CLEAR* -- Clear Variable Storage

CLEAR(STRING,ARGUMENT)

This function causes the values of variables to be set to null. In the simple case, where both arguments are omitted, the action is the same as in BTL SNOBOL-4. I.e. all variables are cleared to contain null. Two extensions are available in SPITBOL. The first argument may be a string which is a list of variable names separated by commas. These represent the names of variables whose value is to be left unchanged. In addition, if a second non-null argument is supplied, then all variables containing pattern values are left unchanged. For example --

```
CLEAR('ABC,CDE,GGG',1)
```

would cause the value of all variables to be cleared to null except for the variables ABC,CDE,GGG and all other variables containing pattern values.

6.9. CODE -- Compile Code

CODE(STRING)

The effect of this function is to convert the argument to type CODE as described in the section on type conversion. The STRING must represent a valid SPITBOL program complete with labels and using ; to separate statements. The call to CODE fails if this condition is not met.

6.10. COLLECT -- Initiate Storage Regeneration

COLLECT(INTEGER)

The COLLECT function forces a garbage collection which retreives unused storage and returns it to the block of available storage. The integer argument represents a minimum number of bytes to be made available. If this amount of storage cannot be obtained, the collect function fails. On successful return, the result is the number of bytes actually obtained.

Note that although the implementation of COLLECT is similar to that in BTL SNOBOL-4, the values obtained will be quite different due to different internal data representations. Furthermore, the internal organization of SPITBOL is such that forcing garbage collections to occur before they are required always increases execution time.

6.11. CONVERT* -- Convert Datatypes

CONVERT(ARGUMENT,STRING)

The returned result is obtained by converting the first argument to the type indicated by the string name of the datatype given as the second argument. The section on type conversion describes the permitted conversions. Any conversions which are not permitted cause failure of the CONVERT call.

Note that SPITBOL does not permit the conversion of all datatypes to STRING. Thus CONVERT(BAL,'STRING') fails rather than giving 'PATTERN' as in BTL SNOBOL-4.

An additional possibility for the second argument is 'NUMERIC', in which case, the argument is converted to INTEGER, REAL or DREAL according to its form.

6.12. COPY* -- Copy Structure

COPY(ARGUMENT)

The COPY function returns a distinct copy of the object which is its argument. This is only useful for arrays, tables and program defined datatypes. Note that SPITBOL does permit the copying of TABLES unlike BTL SNOBOL-4.

6.13. DATA -- Create Datatype

DATA(STRING)

The argument to DATA is a prototype for a new datatype in the form of a function call with arguments. The function name is the name of the new datatype. The 'ARGUMENT' names are names of functions which represent the fields of the new datatype.

Note that in SPITBOL, a significant increase in efficiency is obtained by avoiding the use of duplicate field names for different datatypes, although SPITBOL does allow such multiple use of field function names.

6.14. DATATYPE* -- Obtain Datatype

DATATYPE(ARGUMENT)

DATATYPE returns the formal identification of the datatype of its argument. In SPITBOL, the additional datatype name 'DREAL' is included in the list of possible returned results.

6.15. DATE -- Obtain Date

DATE()

DATE returns an eight character string of the form MM/DD/YY representing the current date.

6.16. DEFINE -- Define a Function

DEFINE(STRING) or DEFINE(STRING,NAME)

The DEFINE function is used to define program defined functions. The use of DEFINE is the same in SPITBOL as in BTL SNOBOL-4.

6.17. DETACH -- Detach I/O Association

DETACH(NAME)

NAME is the name of a variable which has previously been input or output associated. Use of the DETACH function does not affect the file involved.

6.18 DIFFER* -- Test for Arguments Differing

DIFFER(ARGUMENT,ARGUMENT)

DIFFER is a predicate function which fails if its two arguments are identical objects. Note that DIFFER(.ABC,'ABC') succeeds in SPITBOL since .ABC is a NAME. DIFFER and IDENT are the only functions in which the different implementation of the name operator (unary dot) may give rise to problems.

6.19. DUMP* -- Dump Storage

The DUMP function causes a dump of current values. After the dump is complete, execution continues unaffected (the dump function returns the null string). If the argument to DUMP is one, then the dump includes values of all non-constant keywords and all non-null natural variables. If the argument to DUMP is two, then the dump includes values of all array and table elements, and of field values of all program defined datatypes. The format of the latter dump is self explanatory and avoids printing any structure more than once.

A call to DUMP with a zero argument is ignored. This allows use of a switch value which can be turned on and off globally.

6.20. DUPL -- Duplicate String

DUPL(STRING,INTEGER)

DUPL returns a string obtained by duplicating the first (STRING) argument the number of times indicated by the second argument.

6.21. ENDFILE* -- Close file

ENDFILE(STRING)

STRING is the name of a file (not the name of a variable associated with the file). The named file is closed, all associated storage is released and all variables associated with the file are automatically detached. Thus ENDFILE should be used only when no further use is to be made of the file. If the file is to be reread or rewritten, REWIND should be used rather than ENDFILE.

6.22. EQ -- Test for Equal

EQ(NUMERIC,NUMERIC)

EQ is a predicate function which tests whether its two arguments are equal. DREAL arguments are permitted.

6.23. EVAL -- Evaluate Expression

EVAL(EXPRESSION)

EVAL returns the result of evaluating its expression argument. Note that a string can be converted into an expression by compiling it into code. Thus EVAL in SPITBOL is compatible with BTL SNOBOL-4 and handles strings in the same way.

6.24. FIELD -- Get Field Name

FIELD(NAME,INTEGER)

FIELD returns the name of the selected field of the program defined datatype whose name is the first argument. If the second argument is out of range (less than one, or greater than the number of fields), the FIELD function fails.

6.25. GE -- Test for Greater or Equal

GE(NUMERIC,NUMERIC)

GE is a predicate function which tests if the first argument is greater than or equal to the second argument.

6.26. GT -- Test for Greater

GT(NUMERIC,NUMERIC)

GT is a predicate function which tests if the first argument is greater than the second argument.

6.27. IDENT* -- Test for Identical

IDENT(ARGUMENT,ARGUMENT)

IDENT is a predicate function which tests if its two arguments are identical. Note that in SPITBOL, IDENT(.ABC,'ABC') fails since .ABC is a name in SPITBOL. Otherwise IDENT is compatible.

6.28. INPUT* -- Set Input Association

INPUT(NAME,STRING,INTEGER)

The first argument is the name of a variable which is to be input associated. The second argument is the filename of the file to which the variable is to be associated. In OS, the name corresponds to the DDNAME of the file (see section on OS files). If the second argument is omitted, the file name 'SYSIN' (standard input file) is assumed. For compatibility with BTL SNOBOL-4, the second argument may be a one or two digit integer, in which case, the DDNAME FTXXF001 is used. Note however, that the filename 5 is interpreted as SYSIN if no FT05F001 DD card is supplied. Also, there is no provision for multiple files in the FORTRAN sense. Dataset concatenation can be used instead.

The third argument is either zero, in which case it is ignored, or a positive non-zero integer, in which case input records longer than the given limit are truncated.

A restriction in SPITBOL is that only natural variables can be input associated. It is not possible to input associate array and table elements.

6.29. INTEGER* -- Test for Integral

INTEGER(NUMERIC)

INTEGER is a predicate function which tests whether its argument is integral. It fails if the argument cannot be converted to numeric, or if it has a non-integral value.

6.30. ITEM -- Select Array or Table element

ITEM(ARRAY,INTEGER,INTEGER,...) or ITEM(TABLE,ARGUMENT)

ITEM returns the selected array or table element by name. Note that the use of ITEM is unnecessary in SPITBOL because of the extended syntax for array references. (See section on syntax).

6.31. LE -- Test for Less Than or Equal

LE(NUMERIC,NUMERIC)

LE is a predicate function which tests whether the first argument is less than or equal to the second argument.

6.32. LEN -- Generate Specified Length Pattern

LEN(INTEGER) or LEN(EXPRESSION)

LEN generates a pattern which will match any sequence of characters of length given by the argument which must be a non-negative integer. Integer greater than zero.

If the argument is an expression, it is evaluated during pattern matching and must yield a non-negative integer.

6.33. LEQ* -- Test for Lexically Equal

LEQ(STRING,STRING)

LEQ is predicate function which tests whether its arguments are lexically equal. Note that LEQ differs from the IDENT function in that its arguments must be strings, thus LEQ(10,'10') succeeds as does LEQ(.ABC,'ABC').

6.34. LGE* -- Test Lexically Greater or Equal

LGE(STRING,STRING)

LGE is a predicate function which tests whether the first argument is lexically greater than or equal to the second argument.

6.35. LGT -- Test for Lexically Greater

LGT(STRING,STRING)

LGT is a predicate function which tests whether its first string argument is lexically greater than the second string argument.

6.36. LLE* -- Test for Lexically Less or Equal

LLE(STRING,STRING)

LLE is a predicate function which tests whether its first string argument is lexically less than or equal to the second argument.

6.37. LLT* -- Test for Lexically Less

LLT(STRING,STRING)

LLT is predicate function which tests whether its first argument is lexically less than its second argument.

6.38. LNE* -- Test For Lexically Not Equal

LNE(STRING,STRING)

LNE is a predicate function which tests whether its arguments are lexically unequal. LNE differs from the DIFFER function in that its arguments must be strings.

6.39. LOAD* -- Load External Function

LOAD(STRING)

LOAD is used to load an external function. The form of the load argument is the same as in BTL SNOBOL-4 except that the datatype DREAL may be used. In the case where the datatype is unspecified, the form of the descriptor passed is quite different from that in BTL SNOBOL-4. The form of converted arguments is identical. A later document will contain additional details on writing external functions for SPITBOL.

6.40. LOC -- Get Name of Local

LOC(NAME,INTEGER)

The value returned is the name of the indicated local of the function whose name is given by the first argument. LOC fails if the second argument is out of range (less than one, or greater than the number of locals).

6.41. LPAD* -- Left Pad

LPAD(STRING,INTEGER,STRING)

LPAD returns the result obtained by padding out the first argument on the left to the length specified by the second argument, using the pad character supplied by the one character string third argument. If the third argument is null or omitted, a blank is used as the pad character. If the first argument is already long enough or too long, it is returned unchanged. LPAD is useful for constructing columnar output.

6.42. LT -- Test for Less Than

LT(NUMERIC,NUMERIC)

LT is a predicate function which tests whether the first argument is less than the second argument.

6.43. NOTANY -- Build Character Select Pattern

NOTANY(STRING) or NOTANY(EXPRESSION)

NOTANY returns a pattern which will match any single character not in the string argument given. A null argument is not permitted.

If the argument is an expression, then the expression is evaluated at pattern match time and must yield a non-null string.

6.44. OPSYN* -- Equate Functions

OPSYN(NAME,NAME)

The first argument is the name of a function defined to have the same definition as the function named in the second argument. Note that the third argument (allowing redefinition of operators) is not implemented in SPITBOL at this time.

6.45. OUTPUT* -- Set Output Association

OUTPUT(NAME,STRING,STRING)

The first argument is the name of a variable to be output associated. The second argument is the name of the file to which the association is to be made. In OS this is the DDNAME (see section on OS files). If the second argument is omitted, the filename 'SYSPRINT' (standard output print file) is assumed. For compatibility with BTL SNOBOL-4, the file name may be one or two digit integer, in which case the DDNAME FTXXF001 is used. Note however, that the filenames 6,7 are interpreted as SYSPRINT and SYSPUNCH if the corresponding FTXXF001 DD cards are not supplied.

The third argument is the format. It may be entirely omitted. In this case, all parameters are taken from the dataset definition. Strings are transmitted directly. If a string exceeds the specified length (maximum record length for variable length records), then it is split into segments as required.

The second possibility for a format is a single character. This is used for print files. The character given is a control character which is appended to the start of each record. Thus the definition of the standard print file is --

```
OUTPUT(.OUTPUT,,' ')
```

A third possibility for the format is a FORTRAN format. This is supplied for compatibility with BTL SNOBOL-4 and should not be used except where required since format processing is inherently time consuming.

A restriction on the output function in SPITBOL is that only natural variables may be associated. It is not possible to output associate array and table elements.

6.46. POS -- Define Positioning Pattern

POS(INTEGER) or POS(EXPRESSION)

POS returns a pattern which matches the null string after the indicated number of characters has been matched. The argument must be a non-negative integer.

If an expression argument is given it is evaluated during pattern matching and must yield a non-negative integer.

6.47. PROTOTYPE -- Retrieve Prototype

PROTOTYPE(ARRAY) or PROTOTYPE(TABLE)

PROTOTYPE returns the first argument used in the ARRAY or TABLE function call which created the argument.

6.48. REMDR -- Remainder

REMDR(INTEGER,INTEGER)

REMDR returns the remainder of dividing the first argument by the second, the remainder has the same sign as the first argument (quotient).

6.49. REPLACE -- Translate Characters

REPLACE(STRING,STRING,STRING)

REPLACE returns the result of applying the transformations represented by the second and third arguments to the first argument. REPLACE fails if the second and third arguments are unequal in length or null.

6.50. REVERSE* -- Reverse String

REVERSE(STRING)

REVERSE returns the result of reversing its string argument. Thus REVERSE('ABC') = 'CBA'.

6.51 REWIND -- Reposition file

REWIND(STRING)

STRING is the name of an external file (not the name of a variable associated with the file). The named file is repositioned so that the next read or write operation starts at the first record of the file. Existing associations to the file are unaffected.

6.52. RPAD* -- Right Pad

RPAD(STRING,INTEGER,STRING)

RPAD is similar to LPAD except that the padding is done on the right.

6.53 RPOS -- Create Positioning Pattern

RPOS(INTEGER) or RPOS(EXPRESSION)

RPOS creates a pattern which will match null when the indicated number of characters remain to be matched. The integer argument must be non-negative.

If an expression argument is used, it is evaluated during the pattern match and must yield a non-negative integer.

6.54. RTAB -- Create Tabbing Pattern

RTAB(INTEGER) or RTAB(EXPRESSION)

RTAB returns a pattern which matches from the current location up to the point where the indicated number of characters remain to be matched. The argument must be a non-negative integer.

If an expression is used, it is evaluated during pattern matching and must yield a non-negative integer.

6.55. SETEXIT* -- Set Error Exit

SETEXIT(NAME) or SETEXIT()

The use of SETEXIT allows interception of any execution error. The argument to SETEXIT is a label to which control is passed if a subsequent error occurs, providing that the value of the keyword &ERRLIMIT is non-zero. The value of &ERRLIMIT is decremented when the error trap occurs. The SETEXIT call with a null argument causes cancellation of the intercept. A subsequent error will terminate execution as usual with an error message.

The result returned by SETEXIT is the previous intercept setting (i.e., a label name or null if no intercept is set). This can be used to save and restore the SETEXIT conditions in a recursive environment.

The error intercept routine may inspect the error code stored in the keyword &ERRTYPE (see keyword section), and take one of the following actions --

1) Terminate execution by transferring to the special label ABORT. This causes error processing to resume as though no error intercept had been set.

2) Branching to the special label CONTINUE. This causes execution to resume by branching to the failure exit of the statement in error.

3) Continue execution elsewhere by branching to some other section of the program. Note that if the error occurred inside a function, we are still 'down a level'.

The occurrence of an error cancels the error intercept. Thus the error intercept routine must reissue the SETEXIT if required. Examples of error processing routines may be found in the standard SPITBOL diagnostic programs.

6.56. SIZE -- Get String Size

SIZE(STRING)

SIZE returns an integer count of the length of its string argument.

6.57. SPAN -- Create Scanning Pattern

SPAN(STRING) or SPAN(EXPRESSION)

SPAN creates a pattern matching a non-null sequence of characters contained in the first argument which must be a non-null string.

If an expression argument is used, it is evaluated during pattern matching and must yield a non-null string value.

6.58. STOPTR* -- Stop Trace

STOPTR(NAME,STRING)

STOPTR terminates tracing for the name given by the first argument. The second argument designates the respect in which the trace is to be stopped as follows --

'VALUE' or 'V' or null (omitted)	value
'LABEL' or 'L'	label
'FUNCTION' or 'F'	function call and return
'CALL' or 'C'	function call
'RETURN' or 'R'	function return

6.59. SUBSTR* -- Extract Substring

SUBSTR(STRING,INTEGER,INTEGER)

SUBSTR extracts a substring from the first argument, the second argument specifies the first character (1 = start of string), the third argument specifies the number of characters. SUBSTR fails if the sub-string is not a proper sub-string.

6.60. TAB -- Create Tabbing Pattern

TAB(INTEGER) or TAB(EXPRESSION)

TAB creates a pattern which matches from the current position up to the point where the indicated number of characters have been matched. The argument to TAB is a non-negative integer.

If an expression argument is used, it is evaluated during pattern matching and must yield a non-negative integer.

6.61. TABLE* -- Create Table

TABLE(INTEGER)

The TABLE function creates an associative table as in BTL SNOBOL-4. However, in SPITBOL, the table is implemented internally using a hashing algorithm. The integer argument to table is the number of hash headers used. The average number of searches is about M/2N where M is the number of entries in the table, and N is the number of hash headers. Since the overhead for hash headers is small compared to the size of a table element, a useful guide is to use an argument which is an estimate of the number of entries to be stored in the table.

Note that this implementation of table is compatible in that the call used in BTL SNOBOL-4 will work, though possibly not with maximum efficiency.

6.62. TIME -- Get Timer Value

TIME()

TIME returns the integer number of milliseconds of processor time since the start of execution. Note that the values obtained will be different (smaller) than those obtained with BTL SNOBOL-4.

6.63. TRACE* -- Initiate Trace

TRACE(NAME,STRING)

The TRACE function initiates a trace of the item whose name is given by the first argument. The second argument specifies the sense of the trace as follows --

'VALUE' 'V' or null (omitted)	value
'LABEL' or 'L'	label
'FUNCTION' or 'F'	function call and return
'CALL' or 'C'	function call
'RETURN' or 'R'	function return

The following features of the BTL SNOBOL-4 TRACE function are not implemented --

keyword tracing
tracing of array or table elements
program defined trace functions

6.64. TRIM -- Trim Trailing Blanks

TRIM(STRING)

TRIM returns the result of trimming trailing blanks from the argument string.

6.65. UNLOAD* -- Unload Function

UNLOAD(STRING)

String is the name of an external function which is to be unloaded. The restriction in BTL SNOBOL-4 concerning functions OPSYNed to loaded functions does not apply in SPITBOL. A function is not actually unloaded until all functions OPSYNed to it have been unloaded. SPITBOL also allows the names of ordinary functions to appear in calls to UNLOAD. In this case, the result is merely to undefine the function.

7. Keywords

The following is a list of the keywords implemented in SPITBOL. The notation (R) after the name indicates that the keyword is read only, that is, its value may not be modified by assignment.

&ABEND — Normally set to zero. If it set to one when execution terminates, an ABEND dump is given. This is normally used only for system checkout.

&ABORT(R) — Contains the value of the pattern ABORT

&ALPHABET(R) — Contains the 256 characters of the EBCDIC set in their natural collating sequence.

&ANCHOR — Set to zero for unanchored mode and one for anchored pattern matching mode.

&ARB(R) — Contains the value of the pattern ARB

&BAL(R) — Contains the pattern BAL

&CODE — The value in &CODE is used as a system return code if this job is the last in a batch. It is normally set to zero.

&DUMP — The standard value is zero. If the value is zero at the end of execution, then no symbolic dump is given. A value of one gives a dump including values of keywords and natural variables. If the value is two, the dump includes non-null array, table and program defined datatype elements as well. The dump format is self explanatory and deals with the case of branched structures including circular lists.

&ERRTYPE If an execution error is intercepted with the use of the SETEXIT function, then the error code is stored as an integer in &ERRTYPE. The value stored is 1000*majorcode+minorcode. Thus the error code 13.026 is stored as the integer 13026. &ERRTYPE may be assigned a value in which case an immediate error is signalled. This may be useful in signalling program detected errors. If such an error is intercepted, then either the standard error message appropriate to the major code assigned is printed, or a standard message USER ISSUED ERROR MESSAGE is printed if the major code is not in the standard range (1-14).

&ERRLIMIT The maximum number of errors which can be trapped using the SETEXIT function. &ERRLIMIT is initially zero and is decremented each time a SETEXIT trap occurs. SETEXIT has no effect on normal error processing if &ERRLIMIT is zero.

&FAIL(R) Contains the value of the pattern FAIL.

&FENCE(R) Contains the value of the pattern FENCE.

&FNCLEVEL(R) Contains the current function nesting level.

&FTRACE The standard value is zero. If it is set to one, then all function calls and returns are traced.

&FULLSCAN The standard value is zero (QUICKSCAN pattern matching mode). The value is set to one to obtain FULLSCAN mode.

&INPUT Set to one for normal input (standard value). If set to zero, all input associations are ignored.

&LASTNO(R) Contains the number of the last statement executed.

&MAXLNGTH Contains the maximum permitted string length. This value may not exceed 32758.

&OUTPUT Set to one for normal output (standard value). If set to zero, all output associations are ignored.

&REM(R) Contains the pattern REM

&RTNTYPE(R) Contains 'RETURN', 'FRETURN' or 'NRETURN' depending on the type of function return most recently executed.

&STCOUNT(R) The number of statements executed so far.

&STLIMIT(R) The maximum number of statements allowed to be executed. The initial value is 50000. The maximum value allowed is 2**24-1 = 16777215.

&STNO(R) The number of the current statement.

&SUCCEED(R) Contains the pattern SUCCEED.

&TRACE If the value is zero or negative, no trace output is generated. Each line of trace output decrements the value by one. The initial value is 0.

&TRIM Set to zero for normal input mode (standard value). If the value is set to one, all input records are automatically trimmed (trailing blanks removed).

A restriction in SPITBOL is that the only way to change a keyword value is by a direct assignment. Keywords may not appear in any other context requiring a name (for example as the right argument of binary $).

8. Control Cards

Control cards are identified by a minus sign in column one. They may occur anywhere in a source program and take effect when they are encountered. Most of these control card types are special features of SPITBOL and are not implemented in BTL SNOBOL-4.

8.1. Listing Control Cards

Listing control cards are used to alter the appearance of the listing, they have no other effect on the compilation or execution of the program. Listing control cards always occur individually.

8.1.1. -EJECT

The -EJECT control card causes the compilation listing to skip to the top of the next page. The current title and sub-title (if any) are printed at the top of the page.

8.1.2. -SPACE

The -SPACE control card causes spaces to be skipped on the current page. If -SPACE occurs with no operand, then one line is skipped. Alternately, an unsigned integer can be given (separated by at least one space from the -SPACE) which represents the number of lines to be skipped. If there is insufficient space on the current page, -SPACE acts like a -EJECT and the listing is spaced to the top of the next page.

8.1.3. -TITLE

The -TITLE card is used to supply a title for the source program listing. The text of the title is taken from columns 8-72 of the -TITLE card. The subtitle (if any), is cleared to blanks, and an eject to the next page occurs.

8.1.4. -STITL

The -STITL card is used to supply a sub-title for the source program listing. An eject occurs to the top of the next page and the current title (if any) and the newly supplied sub-title are printed. The text for the sub-title is taken from columns 8-72 of the -STITL card. Note that if both title and sub-title are to be changed, then the -TITLE card should precede the -STITL card.

8.2. Option Control Cards

The option control cards allow selection of various compiler options. In each case, there are two modes. Two control cards allow switching from one mode to the other. The mode may be flipped back and forth within a single program. The full names are given for each control card, however, only the first four characters are examined, and the names may thus be abbreviated to four characters. Several control options may be specified on the same control card by separating the names with commas (no intervening spaces should occur). For example

-CODE,LIST,PRINT

In each of the cases listed below, the default option is the one represented by the first of the two control options listed.

8.2.1. -LIST -NOLIST

Normally, the source statements are listed (-LIST option). The -NOLIST option causes suppression of this printout. This may be useful for established programs known to work, or for terminal output. Note that line numbers are always listed on the left which is convenient for terminal output. If compilation errors are detected, the offending statements are printed regardless of the setting of the list mode.

8.2.2. -NOCODE -CODE

The -CODE option causes a printout of the generated code in assembly language type format. This listing may be useful in determining how SPITBOL handles the compilation of various types of statements. The -NOCODE control option resets the normal mode of no code listing. It is permissible to use these cards in combination to obtain listings for selected sections of the source program. The code listing occurs after the end of the source listing starting on a separate page so that the source listing is not affected.

8.2.3. -NOPRINT -PRINT

Normally, control cards are not printed (-NOPRINT). The -PRINT option causes control cards to be listed (provided that the -LIST option is in effect). This option may be useful if serialization is used for updating purposes.

8.2.4. -SINGLE -DOUBLE

The compilation listing is normally single spaced (-SINGLE). The -DOUBLE option causes double spacing to be used, with a blank line between each listed line.

8.2.5. -OPTIMIZE -NOOPTIMIZE

The SPITBOL compiler normally operates in an optimized mode in which the following assumptions are made:

1) The values of BAL, ARB, FENCE, ABORT, REM, FAIL, and SUCCEED are not modified during execution

2) The standard system functions (see section 6 for a full list) are not redefined.

3) Function calls in a statement do not result in modification of values of variables referenced elsewhere in the same statement.

The -NOOPTIMIZE control card specifies that the compiler not make the above assumptions. This results in a higher level of compatability with BTL SNOBOL-4 at the expense of both space and speed. In some cases, the loss of speed may be as much as a factor of ten. The optimizing mode may be switched on and off so that only isolated statements are compiled in non-optimized mode. Note that it is the references to redefined functions which cause the trouble, not the actual definition itself.

8.2.6. -IN72 -IN80

Normally, the compiler reads only columns 1-72 of the input images. Columns 73-80 may be used for serialization. The serialization will be listed on the source listing separated from the program text by a column of dots (this is to prevent accidentally punching past column 72). The -IN80 option causes all 80 columns of the input cards to be read. The -IN72 card resets the normal option. -IN80 should be used from a terminal device, since it eliminates any output on the right side of the page.

8.2.7. -NOSEQUENCE -SEQUENCE

This option is only relevant if -IN72 is in effect. The normal mode (-NOSEQUENCE) ignores any serialization occuring in columns 73-80. If the -SEQUENCE option is taken, then the SPITBOL compiler tests to see whether the serialization is in correct ascending sequence. If an out of sequence card occurs, a message is printed, but no other action is taken (unless -NOERRORS is also specified at the time of the sequence error).

8.2.8. -ERRORS -NOERRORS

Normally execution is allowed even if compilation errors occur (-ERRORS). If a compilation error or a sequence error (-SEQUENCE on) occurs and the -NOERRORS option has been specified, then the execution of the program is suppressed.

8.2.9. -FAIL -NOFAIL

In BTL SNOBOL-4, and in SPITBOL with the -FAIL mode set, a failure in a statement with no conditional goto is ignored and the program execution resumes with the next statement in sequence. This convention often results in errors going undetected, particularly in the case of array references with out of range subscripts and pattern matches which are expected to always succeed. The -NOFAIL option changes this convention. If a statement having no conditional goto is compiled under the -NOFAIL mode, and a failure occurs when the statement is executed, an execution error occurs and a suitable message is generated. The -NOFAIL option is particularly useful for student jobs and other situations where many small programs are being debugged.

8.2.10. -EXECUTE -NOEXECUTE

Normally execution is initiated following compilation. the -NOEXECUTE option, if set at the end of compilation, inhibits execution. This is often useful in conjunction with the option to generate object modules.

8.3. -COPY filename

The -COPY control card allows a section of coding to be copied into the source from an external file. The compiler proceeds as though the text in the file had been read instead of the -COPY card. Filename is any file name which would be legal as the second argument to the INPUT function. In particular, OS allows member names to be supplied in parentheses after the DDNAME which allows sections of code (for example, function definitions), to be stored as members of a partioned dataset. The text copied in may itself contain -COPY cards up to a maximum nesting level of eight levels.

9. Error Messages and Handling

9.1. Compilation Error Messages

When the compiler detects an error, a flag is placed under the point in the statement where the error was discovered and processing of the statement in error is discontinued. Compilation continues with the next statement. Execution is not suppressed unless the -NOERRORS option has been set (see section on control cards). If an attempt is made to execute a statement found erroneous by the compiler, an execution error occurs. Compiler error messages are surrounded by ****** so they are easy to find. The following section describes the various error messages.

******ERROR IN GOTO FIELD******

The goto field is incorrectly formed.

******ERROR IN NUMERIC ITEM******

A numeric item is illegally constructed.

******EXPRESSION IS TOO COMPLICATED FOR THE COMPILER******

The expression being compiled overflows work areas in the SPITBOL. The expression must be broken into two or more statements.

******ILLEGAL CHARACTER******

The compiler detected a character which has no syntactic meaning in the SNOBOL-4 language outside a string literal.

******ILLEGAL TRANSFER ADDRESS******

The operand on an END card is not a simple variable. The operand is ignored and execution starts with the first statement.

******ILLEGAL USE OF , ******

A comma has been used in an illegal context. The only legal uses of comma are to separate array subscripts and function arguments. Note that this error can be caused by accidentally inserting a blank between the function name and the left parenthesis.

******ILLEGAL USE OF < ******

The character < (array left bracket) has been used in a context where an array left bracket cannot legally occur.

******ILLEGAL USE OF) ******

A right parenthesis has been used in an illegal context.

******ILLEGAL USE OF > ******

An array right bracket has been used in an illegal context. This character can be used only to terminate a list or array subscripts.

******ILLEGAL USE OF = ******

An equal sign has been used in an illegal context. Only one equal sign may occur in a statement.

******INVALID -COPY CARD******

A -COPY card has an incorrect filename (this could result from an error in system control card setup), or -COPY has been nested more than eight levels. Compilation proceeds after ignoring the erroneous card.

******LABEL HAS BEEN PREVIOUSLY DEFINED******

The statement has a label which has already been used. Compilation of the statement is discontinued and the earlier definition of the label is retained.

******MISSING END CARD SUPPLIED******

An end of file was read on the system input file (SYSIN) during compilation. The compiler supplies an END card and initiates execution unless the -NOERRORS option is set.

******MISSING OPERAND******

This message is generated when the compiler expects an operand and none is found. For example -- A / / B, (C+)

******MISSING OPERATOR******

The compiler expected an operator and no operator was found. This occurs in situations like (X)A, where an operator is expected after the right parenthesis.

******NON-RECOVERABLE INPUT ERROR******

A non-recoverable input error has been signalled on the system input file (SYSIN). This is a fatal error which terminates compilation and prevents execution. Note that it also cancels any subsequent jobs when a batched run is being processed.

******PROGRAM TOO LONG FOR AVAILABLE STORAGE******

The storage required by the program exceeds available storage. Increase the region allocated and/or the H parameter in the compiler parameter field. Note that storage for execution time use has not yet been allocated. This must be taken into consideration in deciding how much additional memory to allocate. This is a fatal error which terminates compilation and prevents execution.

******UNBALANCED () OR <>******

This occurs if the parentheses or array brackets in a statement are not properly balanced.

******UNDEFINED TRANSFER ADDRESS***

The label used on an END card is not defined. The operand is ignored, and execution starts with the first statement.

******UNMATCHED QUOTE******

A string literal has been started but not properly terminated. Note that string literals cannot be split over continuation cards.

9.2. Execution Error Messages

The execution package performs extensive error checking. When an error is detected, execution is terminated with an error message unless the error is intercepted by means of the SETEXIT function. The message is accompanied by an error code of the form AA.BBB, where AA is the major code and BBB is the minor code. The major code refers to the message given (see below). The minor code further identifies the exact error. The following is a list and explanation of the error messages together with their major codes.

MAJOR = 1 ILLEGAL DATATYPE

In a context where a definite datatype is required, a value of the wrong datatype is given and the attempt to convert it to the correct datatype fails.

MAJOR = 2 UNEXPECTED FAILURE

A statement having no conditional goto failed with the -NOFAIL option set. This usually corresponds to an error such as an unexpected out of range subscript.

MAJOR = 3 ERROR IN ARRAY REFERENCE

An array reference is incorrect. Either the object referenced is not an array or table, or the wrong number of subscripts is given.

MAJOR = 4 COMPILER DETECTED ERROR

An attempt was made to execute a statement found erroneous by the compiler. This message is also issued from statement number 'zero' if compiler errors were detected with the -NOERRORS option set.

MAJOR = 5 ERROR IN REFERENCE TO KEYWORD

An error was made in a keyword reference. Either the operand of & is incorrect, or the value stored is incorrect.

MAJOR = 6 MEMORY OVERFLOW

Dynamic memory is exhausted. Note that this can occur as a result of runaway recursion in function references or pattern matching.

MAJOR = 7 EVALUATION OF GOTO FAILED

If a complex expression is used in the goto field, it is not allowed to fail. Such a failure within a goto expression did occur.

MAJOR = 8 ERROR IN GOTO

The operand of a goto must be a natural variable which is a defined label. Some other value was given. This error message is also given on a return from level zero.

MAJOR = 9 CALL TO UNDEFINED FUNCTION OR OPERATOR

A reference was made to an undefined function, or an undefined operator was used.

MAJOR = 10 ERROR IN ARITHMETIC OPERATION

This message covers a variety of arithmetic errors such as overflow, division by zero etc.

MAJOR = 11 KEYWORD OR SYSTEM LIMIT EXCEEDED

This message is issued when any of the following limits is exceeded -- time, page or card system limits, &MAXLNGTH, &STLIMIT keyword limit.

MAJOR = 12 INPUT/OUTPUT OR OTHER SYSTEM ERROR

An error has been signalled by one of the operating system routines. Some examples are non-recoverable I/O error, load on a nonexistent function etc. Note that the minor codes for this message may differ from operating system to operating system.

MAJOR = 13 INCORRECT VALUE FOR FUNCTION OR OPERATOR

An argument to a function or operand of an operator was of the right datatype, but outside the range of values permitted for some particular use. For example, the null string is an illegal argument for the break function.

MAJOR = 14 VALUE RETURNED WHERE NAME IS REQUIRED

In a context requiring a name (left side of =, goto expression, right argument of $ or .)

9.3. ERROR CODES

This section gives detailed descriptions of the minor codes for all major codes except 12. The latter (system error codes) are system dependent and are given in the following section.

1.001 Evaluated result of deferred argument to POS is not an INTEGER

1.002 Evaluated result of deferred argument to RPOS is not an INTEGER

1.003 Evaluated result of deferred argument to RTAB is not an INTEGER

1.004 Evaluated result of deferred argument to TAB is not an INTEGER

1.005 Evaluated result of deferred argument to LEN is not an INTEGER

1.006 Evaluated result of deferred argument to ANY is not a STRING

1.007 Evaluated result of deferred argument to NOTANY is not a STRING

1.008 Evaluated result of deferred argument to SPAN is not a STRING

1.009 Evaluated result of deferred argument to BREAKX is not a STRING

1.010 Evaluated result of deferred argument to BREAK is not a STRING

1.011 Evaluated result of deferred expression used in a pattern match is not a STRING or PATTERN

1.012 Value to be stored in a keyword is not an INTEGER

1.013 Real argument to loaded function is not a REAL

1.014 Integer argument to loaded function is not an INTEGER

1.015 String argument to loaded function is not a STRING

1.016 Dreal argument to loaded function is not a DREAL

1.017 Operand of unary $ is not a NAME

1.018 Replacing right hand side in a pattern replacement is not a STRING

1.019 Subject of a pattern match is not a STRING

1.020 The pattern in a pattern match is not a PATTERN

1.021 Subscript in reference to one dimensional array is not an INTEGER

1.022 Subscript in reference to a multi-dimensional array is not an INTEGER

1.023 A field function was applied to an inappropriate program defined datatype

1.024 The left operand for alternation or concatenation is not a STRING or PATTERN

1.025 The right operand for alternation or concatenation is not a STRING or PATTERN

1.026 The argument to a field function is not a program defined datatype

1.027 An operand of binary + is non-numeric

1.028 An operand of binary - is non-numeric

1.029 An operand of binary * is non-numeric

1.030 An operand of binary / is non-numeric

1.031 An argument to NE,EQ,LE,GE,LT,GT is non-numeric

1.032 An operand of binary ** is non-numeric

1.033 The operand of unary + is non-numeric

1.034 The operand of unary - is non-numeric

1.035	First argument to LEQ,LNE,LGT,LLT,LGE or LLE is not a STRING
1.036	Second argument to LEQ,LNE,LGT,LLT,LGE or LLE is not a STRING
1.037	Argument to SIZE is not a STRING
1.038	Left operand of binary $ or . is not a PATTERN
1.039	Argument to LEN is not an INTEGER or EXPRESSION
1.040	Argument to POS is not an INTEGER or EXPRESSION
1.041	Argument to TAB is not an INTEGER or EXPRESSION
1.042	Argument to RPOS is not an INTEGER or EXPRESSION
1.043	Argument to RTAB is not an INTEGER or EXPRESSION
1.044	Argument to SPAN is not a STRING or EXPRESSION
1.045	Argument to BREAKX is not a STRING or EXPRESSION
1.046	Argument to BREAK is not a STRING or EXPRESSION
1.047	Argument to NOTANY is not a STRING or EXPRESSION
1.048	Argument to ANY is not a STRING or EXPRESSION
1.049	Argument to VALUE is not a STRING, NAME or correct programmer defined datatype
1.050	Argument to ARBNO is not a PATTERN
1.051	First argument to APPLY is not the name of a function
1.052	First argument to ARG is not a NAME
1.053	Second argument to ARG is not an INTEGER
1.054	First argument to ARRAY is not a STRING
1.055	First argument to CLEAR is not a STRING
1.056	Argument to CODE is not a STRING
1.057	Argument to COLLECT is not an INTEGER

1.058 Second argument to CONVERT is not a STRING

1.059 Argument to DATA is not a STRING

1.060 First argument to DEFINE is not a STRING

1.061 Second argument to DEFINE is non-null and is not the name of a label

1.062 Argument to DETACH is not the name of a natural variable

1.063 Second argument to DUPL is not an INTEGER

1.064 First argument to DUPL is not a STRING

1.065 Argument to ENDFILE is not a STRING

1.066 Argument to EVAL is not an EXPRESSION (or a STRING, which could be converted into an EXPRESSION)

1.067 First argument to FIELD is not a NAME

1.068 Second argument to FIELD is not an INTEGER

1.069 First argument to INPUT is not the name of a natural variable

1.070 File name (second argument) to INPUT is not a STRING

1.071 Format specification (third argument) to INPUT is not an INTEGER

1.072 Argument to LOAD is not a STRING

1.073 First argument to LOC is not a NAME

1.074 Second argument to LOC is not an INTEGER

1.075 Third argument to LPAD is not a STRING

1.076 Second argument to LPAD is not an INTEGER

1.077 First argument to LPAD is not a STRING

1.078 First argument to OPSYN is not the name of a natural variable

1.079 Second argument to OPSYN is not a function name

1.080 First argument to OUTPUT is not the name of a natural variable

1.081	File name (second argument) for OUTPUT function is not a STRING
1.082	Format specification (third argument) for OUTPUT function is not a STRING
1.083	Argument to PROTOTYPE is not an ARRAY or TABLE
1.084	Second argument to REMDR is not an INTEGER
1.085	First argument to REMDR is not an INTEGER
1.086	Third argument to REPLACE is not a STRING
1.087	Second argument to REPLACE is not a STRING
1.088	First argument to REPLACE is not a STRING
1.089	Argument to REVERSE is not a STRING
1.090	Argument to REWIND is not a STRING
1.091	Third argument to RPAD is not a STRING
1.092	Second argument to RPAD is not an INTEGER
1.093	First argument to RPAD is not a STRING
1.094	Argument to SETEXIT is not a label name
1.095	First argument to SUBSTR is not a STRING
1.096	Second argument to SUBSTR is not an INTEGER
1.097	Third argument to SUBSTR is not an INTEGER
1.098	Argument to TABLE is not an INTEGER
1.099	Argument to TRIM is not a STRING
1.100	Argument to UNLOAD is not the name of a function
2.001	Failure of a statement having no conditional goto with -NOFAIL option in effect
3.001	Array reference with one subscript refers to an object which is neither a TABLE nor an ARRAY

3.002	Multi-dimensional array reference refers to an object which is not an array
3.003	Wrong number of subscripts in an array reference
4.001	Compilation errors detected with -NOERRORS option in effect
4.002	Attempted execution of a statement found erroneous by the compiler
5.001	An attempt was made to reference the keyword attribute of a non-natural variable
5.002	Reference to an undefined keyword
5.003	An attempt was made to change the value of a keyword associated with a non-natural variable
5.004	Attempt to change the value of an undefined keyword
5.005	Attempt to change the value of a protected keyword
6.001	Overflow in main dynamic storage area. This can occur as a result of runaway recursion in pattern matching or function reference as well as from generation of too much data.
7.001	The evaluation of a complex goto failed
8.001	RETURN from function level zero
8.002	Transfer to an undefined label
8.003	A transfer to the label CONTINUE occured, but no previous error had been intercepted
8.004	A transfer to the label ABORT occured, but no previous error had been intercepted
8.005	Name used as a goto operand is not the name of a natural variable
9.001	Reference to an undefined function

9.002	Use of the undefined operator -- unary /
9.003	Use of the undefined operator -- binary &
9.004	Use of the undefined operator -- binary ¬
9.005	Use of the undefined operator -- binary @
9.006	Use of the undefined operator -- unary |
9.007	Use of the undefined operator -- unary #
9.008	Use of the undefined operator -- binary #
9.009	Use of the undefined operator -- binary ?
9.010	Use of the undefined operator -- unary %
9.011	Use of the undefined operator -- binary %
10.001	Overflow in + - / or * of two DREALs
10.002	Overflow in + - / or * of two REALs
10.003	REAL division by zero
10.004	DREAL division by zero
10.005	Overflow in REAL ** INTEGER or DREAL ** INTEGER
10.006	Integer division by zero
10.007	Integer addition overflow
10.008	Integer subtraction overflow
10.009	Integer multiplication overflow
10.010	Negative exponent for INTEGER ** INTEGER
10.011	Overflow in integer exponentiation
10.012	DREAL ** DREAL is not permitted
10.013	REAL ** REAL is not permitted
10.014	Integer overflow for unary minus (happens only with largest neg num)

10.015	Attempted division by zero in REMDR function
11.001	Page limit (P parameter) exceeded
11.002	Card limit (C parameter) exceeded
11.003	Input record longer than &MAXLNGTH
11.004	Attempt to set &MAXLNGTH to a value greater than the maximum allowed (32758)
11.005	&STLIMIT set to a value less than the number of statements already executed
11.006	Statement limit (&STLIMIT) exceeded
11.007	Attempt to form a string longer than &MAXLNGTH by concatenation
11.008	A pattern structure has exceeded the maximum permitted size (32K bytes)
11.009	Time limit (T parameter) exceeded
11.010	Attempt to form a string longer than &MAXLNGTH in call to DUPL function
11.011	Attempt to form a string longer than &MAXLNGTH in call to LPAD function
11.012	Attempt to form a string longer than &MAXLNGTH in call to RPAD function
12.XXX	(See section on system error codes)
13.001	Evaluated result of deferred argument to POS is negative
13.002	Evaluated result of deferred argument to RPOS is negative
13.003	Evaluated result of deferred argument to RTAB is negative
13.004	Evaluated result of deferred argument to TAB is negative
13.005	Evaluated result of deferred argument to LEN is negative
13.006	Evaluated result of deferred argument to ANY is null
13.007	Evaluated result of deferred argument to NOTANY is null
13.008	Evaluated result of deferred argument to SPAN is null

13.009	Evaluated result of deferred argument to BREAKX is null
13.010	Evaluated result of deferred argument to BREAK is null
13.011	Operand of unary $ is null
13.012	Argument for LEN is negative
13.013	Argument for POS is negative
13.014	Argument for TAB is negative
13.015	Argument for RPOS is negative
13.016	Argument for RTAB is negative
13.017	SPAN argument is null
13.018	Argument for BREAKX is null
13.019	Argument for BREAK is null
13.020	NOTANY argument is null
13.021	ANY argument is null
13.022	Null first argument in call to the ARRAY function
13.023	An array bound in a call to the ARRAY function is null
13.024	An array bound in a call to the ARRAY function is non-numeric
13.025	In the first argument to ARRAY, a subscript bound has two colons
13.026	An array lower bound in a call to the ARRAY function is not in the range -32768 < LBD < +32768
13.027	An array dimension (HBD-LBD+1) in a call to the ARRAY function is not in the range 0 < DIM < 32768
13.028	Name in CLEAR first argument is null

13.029 Array argument for CONVERT to TABLE is not two dimensional.

13.030 Argument to DATA is null

13.031 Datatype name in argument to DATA is null.

13.032 Missing left paren in DATA argument

13.033 Field name is null in DATA argument

13.034 DATA argument does not end with)

13.035 Too many fields (more than 30), in argument to DATA

13.036 First argument to DEFINE is null

13.037 Function name in first argument to DEFINE is missing (null)

13.038 First argument to DEFINE is missing a left paren

13.039 Argument name in first argument to DEFINE is null

13.040 First argument to DEFINE is missing a)

13.041 Null local name in first argument to DEFINE

13.042 Argument to ENDFILE is null

13.043 Argument to LOAD is null

13.044 Function name in argument to LOAD is null

13.045 Missing (in argument to LOAD

13.046 Missing) in argument to LOAD

13.047 Too many arguments (more than 64) in function to be LOADed

13.048 Argument to REWIND is null

13.049 Argument to TABLE is zero or negative

14.001 A function called by name returned a value

14.002 An expression other than a function call returned a value where a name was required

9.4. SYSTEM ERROR CODES FOR OS/360

This section gives minor codes for major code 12 (system errors) as signalled by the OS/360 interface.

12.001 Invalid file name. Name is too long, or element notation is incorrect

12.002 Missing DD card for referenced file

12.003 Module name for LOAD or UNLOAD is longer than eight characters

12.004 Incorrectable input error

12.005 Incorrectable output error

12.006 Attempt to read past end of file

12.007 Incorrectable input error during loading an external module

12.008 Module for external function not found in library (Possible missing JOBLIB DD card)

12.009 Module to be unloaded is not currently loaded (This is probably an error in the SPITBOL system)

12.010 Attempt to REWIND system file (DDNAME = SYSPRINT, SYSPUNCH, SYSIN)

12.011 Attempt to read from a file previously written on with no intervening REWIND

12.012 Attempt to write on a file previously read from with no intervening REWIND

12.013 Duplication factor or T operand (tab location) in an output (FORTRAN) format is zero

12.014 Illegal character in output (FORTRAN) format

12.015 Too many levels of parentheses in output (FORTRAN) format

12.016 Too many right parentheses in output (FORTRAN) format

12.017 T operand (tab location) is missing in output (FORTRAN) format

12.018 Operand for H (string literal) in output (FORTRAN) format extends beyond the end of the format

12.019 Output format containing more than one character does not start with a left paren and cannot be interpreted as a FORTRAN type format.

12.020 Output format containing more than one character does not start with a right paren and cannot be interpreted as a FORTRAN type format

12.021 The logical record length on the SYSIN file exceeds the maximum permitted (80 characters of data)

12.022 Error in opening file for output

12.023 Error in opening file for input

12.024 Attempt to write two members into the same PDS at the same time. Use ENDFILE to close one of them

12.025 Attempt to reference two files of a magnetic tape at the same time. Use ENDFILE to close old file

10. Programming Notes

The internal organization of SPITBOL is quite different from that in BTL SNOBOL-4. Consequently the relative speed of various operations differs. This section attempts to give some idea of what is going on inside so that the SPITBOL programmer can achieve maximum efficiency.

10.1. Space Considerations

The SPAN, BREAK and BREAKX functions use translate and test tables. For the case of one character arguments, the tables are built into the system and require no additional space. For arguments longer than one character, tables must be built for each call. Each such table requires 260 bytes of storage. If the argument is deferred, no storage is required, but the execution of the pattern is much slower.

ANY and NOTANY allocate 16 byte tables (actually one bit position in a shared 256 byte table)

The space required for each element of an array is 8 bytes in addition to storage required for a string or other structure. All numeric items require no additional space beyond the 8 byte item.

The space required for each non-null element of a table is 24 bytes in addition to space for a string or other structure. A table hash header is 4 bytes. Thus the number of headers can be made reasonably large without using much additional space.

Program defined datatypes require 8(F+1) bytes where F is the number of fields. They are thus quite compact and can be used freely.

The memory required for dynamically compiled code (CODE function) is not reclaimed efficiently in the current version. Improvements will be attempted in future versions.

Each variable block requires 32 bytes. This space is a constant requirement whether or not the variable name has a single use or multiple uses (label,function,variable etc.). This space is never reclaimed once it has been allocated. Thus it is inefficient to use variables to build a table with the $ operator. Instead, use the TABLE datatype.

The COLLECT function can be used to obtain more detailed information on memory utilization for various structures.

10.2. Speed Considerations

To a greater extent than is the case with BTL SNOBOL-4, there is a loss of efficiency in encoding complex structures as strings. Use arrays, tables and program defined datatypes where possible. The latter are particularly efficient in SPITBOL.

A POS pattern may be used freely at the start of a pattern since SPITBOL optimizes this occurence to prevent useless movements of the anchor point. This optimization (which is completely transparent) occurs in both QUICKSCAN and FULLSCAN modes.

Time for datatype conversions will be relatively more noticeable in SPITBOL. Where efficiency is important, avoid unnecessary conversions.

The $ pattern assignment is, if anything, faster than the . pattern assignment and may be used freely.

SPITBOL precomputes all constant expressions before execution. When the OPTIMIZE mode is on (normal case), most patterns can be precomputed, thus no efficiency is lost by writing patterns in line rather than predefining them. Use of the unary * operator to defer computation is still useful in certain cases. For example, consider the following in line pattern matches.

```
X POS(0) ARB N 'X'
X POS(0) ARB *N 'X'
```

The second form is more efficient, since the compiler can precompute the entire pattern.

BREAK, BREAKX and SPAN are very fast, except that deferred arguments having more than one character are quite slow. ARBNO is quite slow.

ARB is slow and should be avoided where possible.

The actual matching process is much faster in FULLSCAN mode than in QUICKSCAN mode since the heuristics require time consuming tests. If a match does not back up much, FULLSCAN may well be faster.

The process of obtaining the value of &LASTNO or &STNO is very slow and these keywords should be used only for debugging.

The SETEXIT error intercepts are fast and may be used for program control as well as debugging.

If a variable is traced or I/O associated, references to the variable are substantially slowed down even if the trace and I/O associations are later removed.

The unary $ (indirect) operator applied to a string argument works differently in SPITBOL and corresponds to a hash search of existing variables. The process of applying $ to a name (including the name of a natural variable) is much faster, which is why SPITBOL returns a name instead of a string when the unary dot (name) operator is used with a natural variable. Thus it is better to use names where possible, for example in passing labels indirectly.

The REPLACE function is optimized when the second argument is &ALPHABET. In this case, the third argument can be used as a translate table directly, and there is no need to construct a table dynamically. The REPLACE function itself can be used to construct the necessary third argument. Thus the call --

```
A = REPLACE(X,Y,Z)
```

may be replaced by the two calls --

```
TBL = REPLACE(&ALPHABET,Y,Z)
A = REPLACE(X,&ALPHABET,TBL)
```

The first of these calls is slow and need only appear once. The second call is fast and could be executed repeatedly for various values of X.

11. OS Data Sets and JCL

11.1. Standard System Files

The following datasets must be defined for operation under OS/360.

11.1.1. DDNAME=SYSIN

This file contains the source images for the compilation and the data. The LRECL on this dataset must not exceed 80 (84 for V format records).

11.1.2. DDNAME=SYSPRINT

This data set is used for printed output including listing of the source program, error messages and trace output. In addition, the standard output variable 'OUTPUT' is associated with the file SYSPRINT. SYSPRINT may be defined with any convenient RECFM and LRECL, except that LRECL=1 is not permitted.

11.1.3. DDNAME=SYSPUNCH

The output variable 'PUNCH' is associated to the file SYSPUNCH by default. Normally this file is used for punched output and is defined accordingly with LRECL=80 (LRECL=84 for V type records).

11.1.4. DDNAME=SYSOBJ

If a DD card is supplied for this file, then SPITBOL will generate an object module for each program compiled. This output is discussed in further detail in section 11.9. If no object module is required, then the DD card for this file should be omitted.

11.2. Additional User Defined Files

The input and output association functions of SPITBOL specify the DDNAME of the file to which the association is to occur. Appropriate DD cards must be supplied for each file used. All RECFM types are supported in SPITBOL.

Filenames may also consist of a DDNAME followed by an element name in parentheses. In the case of a partioned dataset, the element name is a member name. Several members may be read from the same PDS at the same time. However, only one member may be written at a time. To write more than one member, the ENDFILE function must be used to close the previous file.

In the case of a file on tape, the element name in parentheses is an integer file number. Only one file can be opened on a given tape at a time, and again the ENDFILE function must be used to close one file before opening the next.

For variable length record formats, the output string is written as a single record of appropriate length if possible. If the length of the string exceeds the LRECL, then the string is split into several records as required. An oversize record on input causes an error. The null string is written and read as a one byte record consisting of the character X'00' (hexadecimal zero). Spanned records are implemented. However, the input LRECL should not be too much larger than required, since SPITBOL must temporarily find a buffer of length LRECL on input.

For the fixed length record formats, the output string is split into several logical records if its length exceeds the specified LRECL. The last, or only, record written is padded with blanks. This means that on input, extra blanks may be read. A null value is written as a blank record.

For undefined records, the string is written as a block if possible, or split up if necessary. The null string is handled as for variable length records.

It should be clear that variable length record formats are preferable for the input and output of SNOBOL-4 strings, the F formats are implemented primarily for compatibility with other OS/360 processors.

Note that SPITBOL ignores the A (ASA control characters) specification. If control characters are to be generated, the output association should specify an appropriate format. The standard association for SYSPRINT specifies blank control characters.

11.3. Link Editing

The SPITBOL system consists of two assembly modules. The first is the system interface, which has a single CSECT called OSINT. The second assembly module has CSECTS SPITBOLR, SPITBOLC, SPITBOLP, SPITBOLX, SPITBOLA and SPITBOLF.

11.3.1. Single Phase Version

The simplest version is obtained by link editing both modules into a single one-phase structure. This is more efficient than the two phase version and should be used unless memory is severely limited.

11.3.2. Two Phase Version

If memory is limited, about 14K bytes can be saved by linking the compiler and execute packages as separate phases. In this case, we have the following --

Root Phase	OSINT SPITBOLR
Compiler Section	SPITBOLC
Execution Section	SPITBOLP SPITBOLX SPITBOLA SPITBOLF

11.4. Default DCB Parameters

SPITBOL supplies the following DCB parameters if they are omitted

file	default DCB
SYSIN	RECFM=FB,LRECL=80,BLKSIZE=400
SYSPRINT	RECFM=VBA,LRECL=137,BLKSIZE=600
SYSPUNCH	RECFM=VB,LRECL=84,BLKSIZE=172
SYSOBJ	RECFM=FB,LRECL=80,BLKSIZE=400
OTHER FILES	RECFM=VBS,LRECL=2004,BLKSIZE=400

If the RECFM is supplied, then LRECL and BLKSIZE must also be supplied. LRECL and BLKSIZE may be overridden separately. Additional DCB parameters such as OPTCD, BUFNO may be supplied on the DD card as required. Note that certain systems have special requirements for SYSPRINT DCB's. If any difficulty is experienced with printed output, use a DCB which is standard for the installation as copied from one of the cataloged procedures. ASP systems are specially prone to this type of problem.

11.5. Job Batching

Several SPITBOL jobs may be batched together in one run. In this case, SPITBOL need not be reloaded for each program, thus saving considerable system overhead time.

The one phase version is preferable to the two phase version if batching is to be used, since the two phase version requires two extra phase loads for each program in a batch.

To batch programs, follow each program (with its data) with an 'end of file' card having ./* in columns 1-3, except for the last program in the batch which is followed with a normal /* data termination card. See sample deck setup number one for an example.

11.6. PARM Options

The PARM parameter on the EXEC statement may be used to set several parameters for the run. The default setting of the 8 parameters which can be set is given in the following table.

L=16K — Smallest area required for dynamic memory allocation. 16K represents a minimum value. It is unlikely that many programs will execute in less space. The L parameter may be increased for programs known to require more space.

H=100K — Largest region requested for dynamic memory allocation. The default value is set to obtain all available memory. The actual region used is the largest contiguous area available with size between the specified limits. The H parameter may be reduced to prevent SPITBOL from grabbing all available memory.

N=58 — Number of lines per page. This is used for formatting of compilation output. The compiler skips over creases, also -EJECT and -TITLE cards skip to a new page.

D=10 — Maximum number of dumps given. If an internal error abort occurs or if &SYSABEND is set, then a dump is printed on SYSPRINT. Note that this dump is not the standard system dump, but rather is a special format SPITBOL dump. After giving the dump, execution proceeds with the next job in the batch. The D parameter limits the number of dumps which can be given. If an abort occurs after this number of dumps, then a U100 ABEND with the dump option is issued, and a standard system dump will be given if a SYSUDUMP DD card has been supplied.

R=8K — It is necessary to reserve some memory to the operating system for I/O buffers, loaded modules, etc. The R parameters specifies the region to be reserved. This requirement will vary depending on number of files, number of buffers, number and size of loaded functions etc. If too little space is reserved, execution will be terminated with a system error code S80A.

T=5 — Maximum number of seconds of CPU time allowed for compilation and execution of the SPITBOL program. This is independent of the time parameter on the exec statement, which should be set to a higher value so that SPITBOL terminates the run rather than the system.

P=50 — Maximum number of printed pages of output permitted for compilation and execution.

C=0 — Maximum number of punched cards generated (i.e., images on SYSPUNCH).

In a batched run, the parameters T,P,C are applied separately to each program in the batch. This prevents any single program from over-running system limits and aborting the run.

Any or all of these seven parameters may be altered in the PARM field. By using subparameters of the form X=nnn where X is L,H,N,D,R,T,P,C and nnn is the new value. This value may be expressed either as an integer, or in units of 1024 by using a K following an integer. For example --

```
PARM='R=20000,L=16K,T=30'
```

The above PARM specification would reserve 20000 bytes for system use, 16384 bytes for minimum dynamic memory, and allow up to 30 seconds of CPU time. The parameters may occur in any order. For parameters not specified, the appropriate default from the above table is used.

11.7. Region Parameter and Memory Requirements.

The region size required for SPITBOL will depend on the system environment. As a first approximation, use the following --

REGION (one phase version) = 46K + L + R

REGION (two phase version) = 32K + L + R

where L,R are as given in the section on PARM options.

For the standard values of L,R which are minimum values suitable for use with small student programs. These estimates become 70K for the one phase version and 56K for the two phase version.

If the region is too small to obtain the minimum requested dynamic core area, a user ABEND with code U200 is given. If the minimum requested can be obtained but is insufficient, an error message is generated by SPITBOL. This does not terminate processing of further programs if programs are batched together. In either case, the region parameter on the EXEC for SPITBOL must be increased.

If the system area is too small, a system ABEND with code (S80A) will be generated. In this case, the R subparameter in the PARM field should be increased as required.

11.8. User ABEND Codes

The following user ABEND codes can be issued.

CODE	MEANING
100	Internal error abort, send dump to authors
200	Insufficient dynamic memory, increase region
300	Permanent output error on SYSPRINT
400	Missing DD card for system file
500	Error on opening system file

11.9 Linking and Execution of Object Modules

If a DD card for SYSOBJ is supplied, then SPITBOL generates an object module for the compiled program. If several programs are batched together, then more than one object module is generated.

Although these modules are in the standard OS format, it is not possible to link them directly with object modules generated by other system processors or from other SPITBOL runs. Such communication must utilize the LOAD function. The purpose of implementing this feature is to avoid recompilation of programs which are run frequently and also to provide an easy medium for distribution of SPITBOL programs.

In order to execute the object module, it must first be link edited together with the library modules. The control sections in the library modules have similar names to the standard modules and again, it is possible to link in a one phase or two phase version.

To obtain a single phase version, the generated object module is link edited with the library modules. No entry card is required.

To obtain the two phase version, use the following link edit control cards --

```
INCLUDE GMOD  (object module generated by SPITBOL)
INCLUDE LMOD  (library modules)
OVERLAY ALPHA
INSERT SPITBLCL
OVERLAY ALPHA
INSERT SPITBLPL
INSERT SPITBLXL
INSERT SPITBLAL
INSERT SPITBLFL
```

When the resulting program is executed, the same DD cards and PARM field are supplied as for a normal compile and execute run. However, since the program is already compiled, the dynamic memory obtained is used only for execute time data. The SYSIN file should point to the execution data.

11.10 Sample Deck Setups for OS/360

The following is an example of a run where several programs are batched together.

```
// EXEC PGM=SPITBOL
//STEPLIB DD DSN=SPITLIB,DISP=SHR
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD DUMMY
//SYSIN DD *
SOURCE PROGRAM 1
END
DATA FOR PROGRAM 1
./*
SOURCE PROGRAM 2
END
DATA FOR PROGRAM 2
./*
SOURCE PROGRAM 3
END
DATA FOR PROGRAM 3
/*
```

The second example assumes a single program which reads a tape and generates an updated output tape. The program is stored on disk, but the data is supplied from cards. Also the limit parameters are altered to permit 120 seconds of execution, and 2000 pages of output. Note the use of concatenation to allow a program on disk to be concatenated with the input stream. SPITBOL allows concatenation in all cases including concatenation of unlike attributes. The SYSOBJ DD card will cause an object module to be punched on cards.

```
// EXEC PGM=SPITBOL,PARM='T=120,P=2000',
// REGION=140K
//STEPLIB DD DSN=SPITLIB,DISP=SHR
//SYSOBJ DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//TAPEIN DD DSN=MASTER,DISP=OLD,LABEL=(,SL),
// VOL=SER=TP1207  (USE DEFAULT DCB)
//TAPEOUT DD DSN=MASTR2,DISP=(,KEEP),LABEL=(,SL),
// DCB=(RECFM=FB,LRECL=96,BLKSIZE=960),VOL=SER=TP1208
//SYSIN DD DSN=SNOPROG,DISP=SHR
//       DD *
 DATA CARDS
/*
```

Addendum for Version 2.1, 2.2

The description of Version 2.0 applies with the following exceptions:

2.1 Program defined trace functions are available for natural variables.

2.1 OPSYN is permitted for normally undefined operators.

2.4 5. 9) Deferred expressions in pattern matching are assumed to match one character in quickscan mode.

6.11 CONVERT has been changed to allow conversions of all objects to STRING as in BTL SNOBOL4.

6.19 DUMP(3) causes a hexadecimal core dump to be printed.

6.44 OPSYN(NAME,NAME,INTEGER)
OPSYN may be used to redefine operators using a third argument of 1 or 2 as in BTL SNOBOL4 with the following restrictions:

1) Only the first argument can be an operator name.
2) Only normally undefined operators can be redefined.

6.58 An additional second argument 'KEYWORD' or 'K' is used to stop keyword tracing.

6.61 Since the use of even numbers of headers can cause anomolies in the hashing algorithm, TABLE forces its argument odd by incrementing even arguments by one.

6.63 TRACE(NAME,STRING,ARGUMENT,NAME)
Program defined trace functions are available and compatible with BTL SNOBOL4.

Keyword tracing is available for the keywords &STCOUNT, &FNCLEVEL, and &ERRTYPE.

7. &STLIMIT maximum value allowed is 2**31-1

9.1 ******MISSING OPERATOR******
This message is also given when the blanks surrounding a binary operator are omitted.

9.2 MAJOR=4 COMPILER DETECTED ERROR
This message is issued only on an attempt to execute an erroneous statement. The -NOERRORS option will cause generation of the 'EXECUTION SUPRESSED' message following the compilation statistics.

USER ISSUED ERROR MESSAGE
This message is given if &ERRTYPE is assigned a value greater than 14999, or less than 1000.

Addendum for Version 2.1

9.3

4.001	Attempted execution of a statement found erroneous by the compiler.
4.002	Deleted.
8.006	The operand of a direct goto is not code.
9.012	Use of the undefined operator - unary !
13.029	Deleted (this condition causes the CONVERT function to fail).

10.2 References to &LASTNO and &STNO are now reasonably fast.

11.2 Element notation for files (partioned dataset member names, and tape label sequence numbers) is not available.

11.4 The P and C parameters are not implemented.

(Insert for SPITBOL manual version 2.0, S4D23, 2 pages)

SPITBOL NEWSLETTER

#1

December 20, 1971

This is the first issue of what is intended to become a regular newsletter to inform SPITBOL users of current developments and to provide a forum for communication between SPITBOL users. Your comments, suggestions, letters, etc., are solicited for inclusion in future issues.

Robert B.K. Dewar

Kenneth E. Belcher

ILLINOIS INSTITUTE OF TECHNOLOGY

New SPITBOL Release

Version 2.2 of SPITBOL has finally been released. It contains several minor enhancements, and fixes for a large number of bugs as documented elsewhere in this issue. There are currently 29 permanent distribution copies of the SPITBOL system in the field.

Looking Forward to Version 3.0

Version 3.0 will contain several language enhancements including full implementation of trace (allowing tracing of array, table and program defined datatype elements). We are hoping to release version 3.0 in the third quarter of '72. Meanwhile your suggestions for language and implementation enhancements should be sent in our direction. Most of the next issue will be devoted to reporting and discussing suggestions received.

Errors and Restrictions in Version 2.2

Version 2.2 fixes all reported bugs with the following exceptions:

(a) EVAL cannot be called by name,

(b) In -NOLIST mode, some error situations cause the subsequent card to be listed.

These should be regarded as restrictions which will be in force until the release of version 3.0.

JCL for SPITBOL Version 2.0

(a) Move object modules to a PDS

```
//OMOD JOB
// EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSN=TAPE,VOL=SER=TAPE,UNIT=TAPE9,
// LABEL=(1,BLP),DISP=(OLD,PASS),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000,DEN=3)
//SYSUT2 DD DSN=SPITBOL.OBJECT(OSINT)
// DISP=(NEW,CATLG),SPACE=(TRK,(60,10,2)),
// UNIT=2314,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
// EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSN=TAPE,VOL=SER=TAPE,UNIT=TAPE9,
// LABEL=(2,BLP),DISP=(OLD,PASS),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000,DEN=3)
//SYSUT2 DD DSN=SPITBOL.OBJECT(SPITBOL),DISP=MOD
// EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSN=TAPE,VOL=SER=TAPE,UNIT=TAPE9,
// LABEL=(3,BLP),DISP=OLD,
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000,DEN=3)
//SYSUT2 DD DSN=SPITBOL.OBJECT(SPITLIB),DISP=MOD
```

(b) Link edit to load library

```
// EXEC LKED,PARM=(LIST,LET,MAP,XREF,OVLY,NCAL)
//MODS DD DSN=SPITBOL.OBJECT,DISP=SHR
//SYSLMOD DD DSN=SPITBOL.LOAD,DISP=(NEW,CATLG),
// SPACE=(TRK,(100,10,1)),UNIT=2314
//SYSIN DD *
    INCLUDE MODS(OSINT,SPITBOL)      for one phase
    NAME SPITBOL
    INCLUDE MODS(OSINT,SPITBOL      for two phase
    OVERLAY ALPHA
    INSERT SPITBOLC
    OVERLAY ALPHA
    INSERT SPITBOLP
    INSERT SPITBOLX
    INSERT SPITBOLA
    INSERT SPITBOLF
    NAME SPITBOL
    INCLUDE MODS(OSINT,SPITLIB)      for load modules
    NAME SPITPROG
/*
```

(c) Apply superzaps

```
// EXEC PGM=SUPERZAP
//SYSPRINT DD SYSOUT=A
//SYSLIB DD DSN=SPITBOL.LOAD,DISP=OLD
//SYSIN DD *
     (SUPERZAP CARDS AS SUPPLIED)
```

(d) Run (compile and go)

```
// EXEC PGM=SPITBOL,PARM=. . .
//STEPLIB DD DSN=SPITBOL.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(installation values)
//SYSPUNCH DD SYSOUT=A,DCB=(installation values)
//SYSIN DD *
     input
```

(e) Run (create load module)

```
// EXEC PGM=SPITBOL
//STEPLIB DD DSN=SPITBOL.LOAD,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(installation values)
//SYSPUNCH DD DUMMY
//SYSOBJ DD DSN=&MODS,DISP=(NEW,PASS),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
// UNIT=2314
//SYSIN DD *
-NOEXECUTE
     source program
/*
// EXEC LKED,PARM=(MAP,XREF)
//MODS DD DSN=&MODS,DISP=(OLD,DELETE)
//MODL DD DSN=SPITBOL.LOAD,DISP=SHR
//SYSLMOD DD DSN=USERLIB,DISP=OLD
//SYSIN DD *
     INCLUDE MODS
     INCLUDE MODL(SPITPROG)
     NAME PROGNAME
/*
```

New Distribution Format

The new permanent distribution format contains an additional seventh file. This contains source updates to the previous version to obtain the current version (IEBUPDTE format from version 2.1 to version 2.2). DO NOT apply these updates to the 2.2 source.

These updates are provided for the benefit of those interested in seeing what has actually been changed in the source. They may also be of use to those maintaining special system interface modules.

Interesting Techniques Department

The following code implements a generalized page titling facility suitable for inclusion in a variety of programs.

```
        &TRACE = 100000
        DATA('FORMAT(TITLE,LINES_PER_PAGE,LINES_LEFT)')
        DEFINE('PAGE_FORMAT(VAR_NAME,FORMAT)')
        OUTPUT(.TITLE)  :(PFEND)

PAGE_FORMAT  LINES_LEFT(FORMAT)=
.       GT(LINES_LEFT(FORMAT),0)
.       LINES_LEFT(FORMAT)  - 1 :S(RETURN)
        TITLE = TITLE(FORMAT)
        TITLE =
        LINES_LEFT(FORMAT) = LINES_PER_PAGE(FORMAT)
.                   -2    : (RETURN)
PFEND
.  TRACE(.OUTPUT,,FORMAT('1TITLE LINE',56),
.         .PAGE_FORMAT)
          (rest of program)
```

Enhancements in SPITBOL Version 2.2

1. The page and card limit parameters (P,C) have been implemented. The defaults have been set to 100000 to minimize conversion difficulties. Exceeding these limits at execution time causes generation of error codes 11.001,11.002 as described in the manual. Exceeding the page limit during compilation causes termination of compilation with the message ***PAGE LIMIT EXCEEDED***.

2. Time limit exceeded is now recognized during compilation, and causes termination of compilation with an appropriate message.

3. The parameter I=0,1 is added to provide proper processing for precise and imprecise interrupts. The default is I=0 (precise). For use on machines with imprecise interrupts (360/91, 370/195, etc.) I=1 should be used.

*4. The criterion for generation of blank control characters on subsequent lines of a record which is split up has been changed. Blank control characters are supplied if and only if ASA records are involved, regardless of specified OUTPUT format. This solves several reported problems and should represent a better choice of convention.

5. The X'00' convention is no longer used if ASA records are involved. This solves all reported problems with this convention while retaining the useful ability to write null records on read/write files.

6. Generated code for function calls is shorter.

7. Redundant loads of identical constants are avoided.

8. Statements with predicate function calls and no goto field now handle the failure case much faster.

9. The efficiency of unary dot, IDENT, DIFFER, ITEM, APPLY is improved.

10. Commutative binary operators are handled better resulting in a reduction in generated code in some cases.

*11. The handling of string arguments for external functions is now BTL compatible.

12. The name of the LOC function is changed to LOCAL to conform with BTL SNOBOL-4.

Errors in SPITBOL Version 2.1 Corrected in Version 2.2

1. The circuit for printing headings was non-reentrant if VA type record format was used.

*2. The circuit for printing TF messages was in error.

3. The times printed and given by TIME() were incorrect following an intercepted timer overflow or time limit exceeded during compilation.

*4. DATE() gave the wrong date on the first day of the month.

*5. V format records generated garbage characters.

6. One character records for VA, UA format records caused trouble with HASP. An extra blank is added for ASA format records in such cases to avoid control character only records.

7. SPITBOL did not restore the caller's PICA (SPIE). This is a violation of OS standards and caused storage use problems in some environments.

8. An incorrect base register setting could cause trouble with CODE/EVAL calls.

9. Syntax errors in CODE/EVAL arguments could cause subsequent regeneration failures.

*10. Pattern matches with a constant subject and constant pattern compiled erroneously.

11. Use of a constant as a goto operand (:F(3)) caused execution failure.

12. Statements with complex left hand sides erroneously evaluated the right hand side first.

*13. Output association was not checked in the case of left hand side function calls.

14. Complex goto fields could leave a base register incorrectly set.

*15. CODE, EVAL arguments ending with a variable name were handled incorrectly.

*16. The character X'00' could cause compiler abort.

*17. Incorrect code generation for unary * when used as a function argument and in other complex expressions.

18. The statement = value (missing left side) caused compiler abort.

19. Constants on the left side of an = were modified as indicated, no error being reported.

20. The use of a constant as a right argument to binary dollar or dot (e.g. X $ 5) was not detected as an error.

21. Calls to INTEGER and BREAKX were not recognized by the -CODE lister.

22. SYSIN records longer than 80 characters (from -COPY cards) caused compiler malfunction. They are now treated as I/O errors.

23. Sequence error messages were not always listed in -NOLIST mode.

*24. All blank strings were converted to integer zero. This should cause a conversion error.

*25. The presence of leading and trailing blanks in a string to be converted to an integer left a base register set incorrectly, resulting in subsequent failure.

26. Strings with one leading and/or trailing blank failed on attempted conversion to integer.

27. The use of $ to create new variable blocks in programs using the OUTPUT function could cause failure.

28. There is a limit on number of variable blocks allowed (about 3-4K). Exceeding this limit caused system failure. Now, error code 6.002 is issued.

*29. SPAN with one character deferred argument failed.

*30. Several pattern primitives could fail in QUICKSCAN mode when used within a recursive match (unnecessary rematches attempted).

*31. Useless matching of alternatives attempted in some cases in QUICKSCAN mode.

*32. Overtime not recognized in certain long pattern match loops.

33. The (strange) construction: X . *V caused an abort.

*34. Stack temporaries not relocated in garbage collection.

35. Like 34, but rarer and <u>not</u> fixed in TF's.

36. Garbage collection during pattern construction could cause fatal error in collector.

*37. Undefined operator table not relocated.

*38. Miscellaneous other system fields not relocated.

39. Incorrect garbage collect call from unary $ function.

40. Incorrect garbage collect call from processing of INPUT associated read calls.

*41. Handling of long strings incorrect in calls to LOAD'ed functions.

42. The use of :(RETURN) in a SETEXIT routine activated by an error during pattern matching or expression evaluation caused an abort.

*43. Pattern matches with a string pattern could match erroneously in certain unusual cases.

*44. Table elements could be lost in certain cases or entered more than once.

45. Real overflow or division by zero could give the wrong statement number in the message or erroneous action if intercepted.

*46. Incorrect &LASTNO value when last statement executed contained an expression evaluation.

47. Statement count trace could cause errors in large programs if the trace used a fourth argument.

48. Number of statements executed printed incorrectly when it should be zero.

49. Errors occuring in constant expressions could not be intercepted more than once.

50. A memory overflow during evaluation of constant expressions was not correctly recognized. (This could lead to generation of object modules with inherent memory overflows.)

*51. Error code 14.001 was issued incorrectly (the statement number could be wrong).

*52. EVAL incorrectly re-evaluated results which were expressions.

53. APPLY could bomb when used with system functions.

*54. ARG failed instead of issuing error code 1.052 for undefined functions.

55. Conversion of NAMEs to NAME failed.

56. Conversion of STRINGs to NAME gave an erroneous result (in CONVERT call).

57. Conversion of NAMEs to INTEGER gave unrepeatable erroneous results.

*58. EVAL function gave an error code instead of failing for bad arguments.

*59. LPAD with a null first argument could give wrong results.

*60. LPAD with second argument > 255 gave wrong results.

*61. RPAD with second argument > 255 gave wrong results.

*62. DUPL could be used to generate strings longer than &MAXLNGTH.

*63. TRACE malfunctioned when a fourth argument was given.

*64. VALUE could not be called by name.

*65. VALUE did not accept NAME's as arguments.

*66. REWIND function caused an error abort.

* Corrected by 2.1 SUPERZAPS.

Errors Fixed in TF 2.2.1

1. Failure to make proper test for storage overflow when creating new variables at execution time (via unary $). Fatal effect in some cases.

2. Error in TF printing circuit (again!).

SPITBOL NEWSLETTER

#2

April 1, 1972

Robert B.K. Dewar
Kenneth E. Belcher
ILLINOIS INSTITUTE OF TECHNOLOGY

Errors corrected by TF 2.2.3

1. Null records written to UA or VA type files still written as X'00' instead of blank.
2. Page limit exceeded during code listing causes a bomb.
3. Erroneous pipe line drain in numeric conversion.
4. Division by 0 when I=1 (imprecise interrupts) causes abort.
5. STOPTR operates incorrectly, possibly causing an abort.
6. Function tracing (via TRACE) not operative in certain cases.
7. EVAL compiles concatenation as a pattern match in some cases.
8. Incorrect garbage collect call on return from external function which has a string value. Usually causes an abort.
9. Exponent overflow (U100 abort) during attempt to convert out of range reals.
10. Keyword trace on &STCOUNT terminates if &STLIMIT is changed.
11. Incorrect compilation of statements referencing &LASTINO when I=1.
12. Dump after error 8.001 causes U100 abort.
13. Tables created by conversion from array have an even number of hash headers.
14. The binary dot operator assigns garbage values in complicated pattern matches.
15. The attempt to convert long strings with leading blanks to integer causes an abort.
16. The attempt to compile over-complicated expressions with CODE or EVAL can cause an abort instead of a compilation failure.
17. The error check to prohibit input records longer than 80 characters is inoperable, leading to an abort on the attempt to read long records as source input to the compiler.

18. Pattern elements of the form *variable where variable contains an expression give an erroneous message.

19. Evaluation of an EXPRESSION which is an input associated variable causes an abort.

FR Status Listing

All FR's are identified as follows:

V.R.C.N

V version number

R release number

C copy number

N serial number (from FR form)

The disposition codes for closed out FR's are as follows:

Fn error fixed by TF V.R.n

S filed as a suggestion, not a bug

D fix deferred to next source release

N not a bug (misunderstandings, etc.)

U unsolved problem, insufficient data

E system error (not a SPITBOL problem)

Closed FR's

FR ID	Disposition	Comments
2.2.0002.9	F3	
2.2.0004.1	S	Request to alter SPITBOL to work with page 0 fetch protected.
2.2.0004.2	F3	

FR ID	Disposition	Comments
2.2.0016.RR1	D	Strings with a large number of leading zeros cannot be converted to integers.
2.2.0017.73	F2	
2.2.0017.74	S	Currently the error codes 1.017, 13.011 can arise from NRETURN processing. The suggestion calls for separate codes.
2.2.0017.75	F2	
2.2.0017.76	F2	
2.2.0017.77	F3	
2.2.0017.79	F3	
2.2.0017.80	N	
2.2.0017.81	F2	
2.2.0017.82	N	
2.2.0017.83	F3	
2.2.0017.84	F3	
2.2.0017.85	F3	
2.2.0017.86	F3	
2.2.0017.87	S	A continue after exceeding &STLIMIT is treated specially: the statement is not failed. The suggestion, which is not possible to implement, is to treat other errors similarly.
2.2.0017.88	S	Long strings of digits are converted to REAL in BTL SNOBOL4. The conversion fails in SPITBOL.
2.2.0017.89	F3	
2.2.0017.90	S	Include prototypes in string representation of TABLE's and ARRAY's.

FR ID	Disposition	Comments
2.2.0017.91	S	Allow second argument (library name) in LOAD call.
2.2.0017.92	D	Provide uniform recovery from failure in success goto.
2.2.0017.93	U	Unidentified U100 abort.
2.2.0017.94	F3	
2.2.0017.95	F3	
2.2.0017.96	F3	
2.2.0017.97	F3	
2.2.0017.98	F3	
2.2.0017.99	E	Grabage characters on last line using ASP.
2.2.0017.103	F3	
2.2.0019.69	F3	
2.2T003.1	N	

Open FR's

Open FR's are problems which are unsolved but still being worked on:

2.2.0017.78	Infinite loop after overtime from succeed/fail type loop.
2.2.0017.100	Specification exception (u100) during EVAL call.
2.2.0017.101	Erroneous 6.001 error with memory available.
2.2.0017.102	Loop caused by erroneous use of CONTINUE.

1108 SPITBOL

We have reached the debugging phase, with many features already operational, such as tracing, programmer defined functions, and integer arithmetic. It appears to be faster than SPITBOL on a 360/65, as should be expected. The system is about the same size (in characters) as 360 SPITBOL, and operates in quarter word mode, using 8 bit ASCII internally. The interface will permit I/O in FLD, EBCDIC, or ASCII. The implementation encompasses some promised 360 enhancements, such as tracing on all names, including arrays, tables, etc., plus a cross referencing feature. It is expected that &STFCOUNT will be implemented, along with user-written-code overlays in a transparent fashion.

We hope to have a locally operational version in about a month, and perhaps a month beyond that to prepare a distributable version.

External Functions in SPITBOL (Version 2)

This section gives detailed information necessary for writing external functions in FORTRAN and assembly language for use in SPITBOL. To a great extent, compatability with BTL SNOBOL4 is maintained but there are some differences.

An external function exists as a load module in one of the standard job libraries (SYS1.LINKLIB, JOBLIB or STEPLIB). Most usually, STEPLIB is pointed to a private library containing the module. Concatenation may be used to introduce more than one library if required. Note carefully that the module name is the same as the entry name and the function name.

The function is introduced by means of the LOAD system function which is compatible with that supplied in BTL SNOBOL4 with minor exceptions:

LOAD('fname(arg 1, arg 2, . . . arg N) result')

fname — is the function (and module) name. It must not be longer than 8 characters (there is no truncation of longer names as in BTL SNOBOL4).

arg 1, arg 2 . . — are the argument datatypes which may be STRING, INTEGER, REAL, DREAL. Any other entry (including null) means that no conversion takes place. If one of these four special entries is used the corresponding argument is converted to the indicated datatype and passed in special external form.

result — is the result type specified in a similar manner. It may be omitted if no conversion is required.

Note that in SPITBOL, LOAD does not permit a second argument at the current time (version 2).

The call to the function obeys standard OS360 conventions:

(1) points to parameter list
(13) points to save area
(14) return address
(15) function entry address

Also register (8) points to the SPITBOL data area for use by functions which interact in an intimate way with the SPITBOL system.

In accordance with OS standards, any registers used must be saved and restored in the save area. (BTL SNOBOL4 allows destruction of registers).

Parameters are given as addresses of double word quantities as follows:

STRING	Word 1	starting address
	Word 2	length in bytes
INTEGER	Word 1	integer value
	Word 2	unused
REAL	Word 1	real value
	Word 2	unused
DREAL	Words 1, 2	long form real value
Unconverted	Words 1, 2	standard SPITBOL value specifier, see SPITBOL listing for details.

It should be noted that the form of unconverted arguments is totally incompatible with BTL SNOBOL4. STRING's are not alligned on any given boundary. Note that the converted forms of numeric data are suitable for use in a call to a FORTRAN function.

The result is returned as follows:

INTEGER	(GR0)	has integer value
REAL, DREAL	(GR0)	has real or long real value
STRING	(GR0)	points to two word block with word 1 = address, word 2 = length in bytes.
Unconverted	(GR0)	points to eight byte SPITBOL value specifier.

To return to SPITBOL, execute

```
BR 14   (success return)
B 4(,14)(failure return)
```

External Data Types

Special external data types may be introduced as special 8 byte specifiers passed unconverted with the first byte set to one of the following values:

X'20', X'22', X'24', X'26', X'28', X'2A', X'2C', X'2E'

Such external data only has significance to external functions which recognize it.

UNLOAD Function

The call UNLOAD(.fname) has the effect in SPITBOL of undefining the function fname for any kind of function. SPITBOL automatically removes an external function from storage if all functions referring to it become undefined (or redefined). Thus in the normal case, UNLOAD is compatible with BTL SNOBOL4, but is also works in complex uses of OPSYN and function redefinition, avoiding bugs which are currently present in BTL SNOBOL4.

Suggestions

This section contains a series of miscellaneous suggestions for improvements, changes, etc.. The contributors are not individually identified since many suggestions have come from more than one direction, and many stem directly from our work. However, we would like to take special note of the many suggestions from Richard Siegler (Columbia University), who also designed the suggestion form which we have adapted as standard.

Not included here are suggestions for efficiency optimizations. The following general policy is in effect:

Space is a vital factor. The system will not be made larger to decrease running time unless the gains are really substantial (i.e. more than 10% in a significant number of programs).

Now here are the suggestions. Please react to them, especially where the changes might impact your use of the system.

1. Provide a facility for labels which are not stored in the symbol table (to save space). Such labels could not be referenced indirectly, traced or used as function entry points. Two ways of doing this:

   (a) A control card listing either labels to be treated this way or those not to be treated this way. Absense of control card implies all labels to be handled as at present.

   (b) Special first character in label (eg. %) indicates this usuage.

This could save approximately 35 bytes/label.

2. New keywords, many possibilities:

&CALLTYPE	'NAME' or 'VALUE' depending on how the current function was called (note: could not be implemented in BTL SNOBOL4 environment).
&TIME, &PAGES, etc.	Values of various system job limits for monitoring purposes.
&PATLEVEL	Incremented during recursive pattern matching. Useful for dynamic patterns.
&NEWTRACE	Allow program defined trace functions to change saved value of &TRACE.

&PTRACE	Activate pattern match trace. (What should the output be?)
&OPT	Control optimization of code generated at execution.
&VERSION	Identify implementation version.
&COMPNO	Current next statement number to be compiled.
&COMPERR	Identify type of compilation error in CODE or EVAL call.

3. Remove -FAIL, -NOFAIL feature. We consider the introduction of this to be an error. Are you using it?

4. Do away with chronological ordering of table elements when converted to array. (This feature exists in BTL SNOBOL4 but is not documented). Saves time and space (garbage collector could remove null elements from a table).

5. New pattern function (quite difficult to do, but certainly useful):

AND(P1,P2)	matches strings matched by P1 and P2.
NOT(P1)	matches null unless P1 would match in which case it fails.

6. Provide a mechanism for assignments during pattern matches. This can be done now by *ASSIGN(.var,value). A special feature would take quite a bit of space for a small gain in speed which probably violates the optimization policy.

7. Extend DUMP function to work with specific objects.

8. Two new functions:

LABSYN(.L1,.L2)	copies a label definition attribute.
LABCLEAR(code)	undefines all labels references by a code object.

9. Evaluate right hand sides before left hand sides in non pattern match statements in optimized mode. Version 2.1 used to do this, no one even noticed let alone complained. Fixing this "bug" can noticeably slow down programs with statements such as:

```
T<x> = IDENT(J) 3
```

10. Make :(CONTINUE) for return from &ERRTYPE trace always cause statement failure. This would be much cleaner than the current local failure convention.

11. Provide functions and perhaps a new datatype for bit manipulations (OR, AND, XOR, etc.).

12. Implement random access I/O, including ISAM type support. One possibility would be to have the files logically large arrays (or tables for ISAM), with access to a certain element such as FILE<1> or ISAM<'ABC'> causing the appropriate I/O.

13. Cross referencing of variable references and perhaps inherent errors that don't produce compilation diagnostics, such as 1 = 2.

14. Provide more information on space utilization, perhaps with a traceable keyword (&SPACE?) and/or information with a dump on space utilized for each structure.

15. Provide array, table, etc., tracing on all elements by a single trace call, rather than having to trace each element individually.

16. DO NOT EMBELLISH. (suggestion from R. Griswold). This is an excellent suggestion to be borne in mind when reading the above list or supplying new suggestions.

SPITBOL NEWSLETTER

#3

June 16, 1972

Robert B.K. Dewar

Kenneth E. Belcher

ILLINOIS INSTITUTE OF TECHNOLOGY

Table of Contents

Errors Corrected by TF 2.2.4

1. Garbage collection following an intercepted error in pattern matching can retain an unneeded garbage item.

2. T parameter value > 300,000 causes an abort.

3. Garbage collection during programmer defined trace of a function causes an abort.

4. EVAL may loop doing endless garbage collects if core is tight.

5. Garbage collection during a CODE call may cause an abort.

6. Object module reload fails to relocate strings which have been converted from small integers during preevaluation of constants. (This is unusual, an example is the pattern 1 $ A $ B).

7. Erroneous garbage collect call from string concatenation (consequences are obscure, may not cause any trouble).

8. Use of input associated variables in other than simple assignment statements may give erroneous results.

9. The use of a control card with some other option following IN72 or IN80 causes an abort.

10. Changing &STLIMIT resets &STCOUNT to zero. (Inadvertantly introduced in TF 2.2.3 which must be applied prior to TF 2.2.4).

11. Attempt to intercept the error caused by an erroneous branch to CONTINUE (code 8.003) causes an abort.

12. Abort may result from creating a new variable block with a call to a function such as DEFINE or DATA.

13. Memory overflow (code 6.001) may be signalled with memory available when creating a new variable block.

14. Miscellaneous destruction of stacks, data, etc., can occur if CODE or EVAL is used when core is tight.

15. Overtime during pattern match causes an abort.

FR Status Listing

All FR's are identified as follows:

V.R.C.N

V Version Number
R Release Number
C Copy Number
N Serial Number (from FR form, starts with NS if submitted on non-standard form)

The disposition codes for closed out FR's are as follows:

Fn error fixed by TF TF V.R.n
S filed as a suggestion, not a bug
D fix deferred to next source release
N not a bug (misunderstandings, etc.)
U unsolved problem, insufficient data
E system error (not a SPITBOL problem)

Closed FR's

FR ID	Disposition	Comments
2.2.0002.NS1	F4	
2.2.0002.NS2	F4	
2.2.0005.NS1	F4	
2.2.0005.NS2	N	
2.2.0009.NS1	F4	
2.2.0016.NS1	E	Abort on changing &STLIMIT (OS R21 on 370/165)
2.2.0017.78	F4	
2.2.0017.100	F4	
2.2.0017.101	F4	
2.2.0017.102	F4	
2.2.0017.104	F4	

FR ID	Disposition	Comments
2.2.0017.105	F4	
2.2.0017.106	F3	
2.2.0017.107	F4	
2.2.0017.108	F4	
2.2.0017.109	F4	
2.2.0017.110	F4	
2.2.0017.111	F4	
2.2.0017.112	F4	
2.2.0019.67	F2	
2.2.0019.70	F4	
2.2.0020.NS1	F4	
2.2.0033.NS1	F4	
2.2.0039.001	F4	

Open FR's

2.2.0019.201	Erroneous results, more information required (may possibly be fixed by TF 2.2.4).

1108 SPITBOL

Debugging is proceeding smoothly. Initial distribution is scheduled for October 1, 1972.

360/370 DOS SPITBOL

We now have available a version of SPITBOL running under DOS. This version includes a comprehensive file handling interface which can handle any sequential format on disks or tapes. Distribution of the DOS version which can run on a 128K machine will start October 1, 1972.

SPECTRA/70 SPITBOL

A version of SPITBOL for the UNIVAC(!) Spectra/70 is being prepared and will hopefully be ready later this year. (The sample programs have run successfully.)

ICL 1900 SPITBOL

A version of SPITBOL for the ICL 1900 series computers is expected to be completed and available later this year. Of particular interest is the fact that this version is largely coded in machine independent macros and should be transferable to other machines later on.

External Functions Revisited (See Newsletter #2)

Some additional corrections and clarifications:

REAL and DREAL results appear in FRO (not GRO!).

String results are always copied to free core on return and may therefore be built inside the function.

The SPIE active is one which generates a dump if an interrupt occurs which is unknown to SPITBOL.

The Program mask is all zeros on entry and must be all zeros on exit.

For those who know the innards of SPITBOL, many functions of the resident execution package are available to an external function. At the current time, there are no plans to document such facilities beyond the documentation in the SPITBOL listing itself, since these facilities are subject to change.

Suggestions - The Old

Last issue's suggestions did draw some mail and the following is a further examination incorporating user comments.

1. (Transparent labels)

Mixed Reactions. This is clearly 'clutter' in the interests of efficiency and as such should be resisted unless overwhelmingly necessary. Also describing which labels must be in the symbol table is awkward (e.g. function entry points). On balance we are inclined to reject this. (See also new suggestion number 4.)

2. (New Keywords)

&CALLTYPE	This needs more explanation: F(X) = G(A) within F, &CALLTYPE = 'NAME', and within G, &CALLTYPE = 'VALUE'. On balance, this should probably be rejected due to difficulties of implementation in some schemes.
&TIME, &PAGES, etc.	Probably not worthwhile.
&PATLEVEL, &NEWTRACE	Useful and easy to implement.
&PTRACE	Defer this pending some viable suggestions for implementation.
&OPT	Need this really be dynamic? Why not use exit value from main compilation.
&VERSION	?
&COMPERR, &COMPNO	See New Suggestions.

3. (-NOFAIL removal)

User reactions: "horrors", "don't understand". OK, we changed our mind. NOFAIL is a good feature and we were clever to think of it the first place.

4. (Chronological ordering of Table Elements)

This will be removed. Order will be "undefined".

5. (AND, NOT)

These are useful and were positively received. However, on further reflection, there is a need for four functions as follows:

AND(A,B)	Matches strings matched by A and B
NOT(A)	Matches any strings not matched by A and extends like ARB.
FAIL(A)	Matches null unless A would match in which case it matches null.
NULL(A)	Matches null if A would match else it fails.

For an example of the last two, consider these replacements:

```
BREAK(A) = ARBNO(NOTANY(A))  NULL(ANY(A))

SPAN(A)  = ARBNO(ANY(A))  ANY(A)  FAIL(ANY(A))
```

To get a feeling for these, try writing SPAN and BREAK without these functions.

There is some argument that NOT should be replaced by:

ANDNOT(A,B)	Matches strings matched by A but not matched by B.

This is because NOT as it stands is an implied ANDNOT(ARB,A). That is: an ARB is used to provide the alternatives for testing against A which may be very slow.

For example:

```
AND(LEN(3),NOT('ABC'))
```

may be expected to be much slower than

```
ANDNOT(LEN(3),'ABC')
```

since the former is equivalent to

```
AND(LEN(3),ANDNOT(ARB,'ABC'))
```

6. (Pattern dynamic assignments)

Practically no gain in speed, not worthwhile (especially if = is made into an operator).

7. (DUMP extension)

Further explanation:

```
DUMP(2,A)
```

Where A is an array would print the array and all its linked contents.

8. (LABSYN, LABCLEAR)

Seem worthwhile if they can be done.

9. (Right side first on assignment)

Probably unwise, especially if = is made an operator (see New Suggestions).

10. (statement failure for CONTINUE)

This is an improvement and will be done.

11. (Bit String Manipulations)

This appears worthwhile. Here is a specific scheme:

Datatype name:	BITS	
Constant denotation:	B'10101' etc. (maybe X'324' and/or O'177' as well.)	
Functions:	BAND, BOR, BXOR, BCOMPL, BSHIFT, BSUBSTR	
Operators:	concatenation	
Conversions:	BITS → STRING	String of '0's and '1's of same length as number of bits.
	BITS → INTEGER	Right adjust in a word and truncate to word lenth excluding sign bit. Error if non-zero bits dropped.
	BITS → (D)REAL	Convert to integer first.
	STRING → BITS	Bit string of same number of characters. Error if not all '0's and '1's.
	INTEGER → BITS	Error if not positive. Else minimum length bitstring representing number in binary.
	(D)REAL → BITS	Convert to integer first.

In addition there need to be two (implementation dependent) functions for mucking with internal representations:

UNSPEC(arg)

Where 'arg' is an INTEGER, STRING, REAL or DREAL yields the bit string corresponding to the internal representation.

SPEC(bits, type)

Yields a value of datatype 'type' (INTEGER, STRING, REAL, DREAL) using 'bits' as the bit pattern.

(Thanks to J.H. Lindsay, Queen's University for some helpful suggestions here.)

12. (Direct Access)

Possible details:

FARRAY(filename)	Yields an object of datatype ARRAY whose elements are fixed length strings.
FTABLE(filename)	Yields an object of datatype TABLE whose elements are variable length strings. The subscript names correspond to record keys.

The above files can also be processed sequentially by using the normal I/O procedures in conjunction with two special functions:

NOTE(filename)	Yields the identification (block number or record key) of the next record to be processed.
POINT(filename, id)	Sets the identification of the next record to be processed.

13. (Cross-reference)

This will be added if possible. It is not likely that this feature would be extended to dynamically compiled code as one user suggested.

14. (Space Utilization Statistics)

Will be provided is possible.

15. (Array trace with single call)

    Probably not (causes trouble with other implementations).

16. (DO NOT EMBELLISH)

    AMEN said Dr. Roosen-Runge (Toronto).

And the New

1. Make = an operator which could occur anywhere. It would be a low priority binary operator returning the assigned variable or the assigned value as appropriate.

2. Introduce ? as a binary pattern match operator (subject ? pattern). ? could occur anywhere and would yield the matched substring as value. It would also be possible to use this on the left side of an assignment with the usual result of replacement. ? could be omitted in the outer position to retain compatability.

3. Allow arbitrary sequences of operator symbols to be used as binary operators (with OPSYN). This causes no ambiguity and, in fact, neatly deals with the case of '**'. Priority could be that of the first symbol or some standard default.

Note that 1 - 3 are actually implemented in SITBOL, a SNOBOL-4 implementation for the PDP10 by J. Gimpel, Bell Labs, Holmdel, New Jersey.

4. Provide some kind of IF-THEN-ELSE structure. This is desparately needed. One suggestion:

```
if      statement;   OR   if   statement;
then    statement;        then statement;
        statement;                  .
        statement;                  .
            .                       .
            .                       .
        statement;                statement fi;
else    statement
            .
            .
            .
        statement fi;
```

Where if, then, etc. are keywords either reserved, or written in some special way (e.g. IF, with following comma) which would be syntactically distinct. Other similar structures such as case, for, while, until, could be provided. The boolean condition tested is success/failure of the statement.

5. Use ; as a statement terminator consistently (as in PL/1, ALGOL-60, ALGOL-68) and do away with continuation characters.

6. Allow labels to be indicated by a colon (no confusion need arise with GOTO's which are always preceeded by a blank).

Suggestions 5, 6 (or 4, 5, 6) could be implemented under a control card switch to ensure compatability.

7. Provide the following keywords:

&ERRMSG	Text of error message following execution error or after a compilation failure in CODE or EVAL.
&COMPNO	Number of next statement to be compiled. (Can be modified.)
&ERRPOS	Scan pointer position in string after CODE or EVAL compilation failure.

These three keywords would allow the compiler to be written in SPITBOL. This would make possible a fully conversational system in a TSO or other similar environment.

The One Character Question

The one character assumption for expression patterns which is now implemented (see S4D23 addundum at the back of the manual) causes trouble in some cases. Should it be changed? Send in your vote today.

SPITBOL NEWSLETTER

#4

March 2, 1973

Robert B.K. Dewar
Kenneth E. Belcher
Illinois Institute of Technology

Table of Contents

Errors Corrected by TF 2.2.5

1. In some cases, running out of room at compile time causes an abort instead of an error message. Also this out of room condition can be signalled falsely.

2. Concatenation of long strings can cause an erroneous result from an incorrect storage regeneration.

3. If an input statement causes a garbage collection, miscellaneous destruction may occur. We have seen several cases of this error which was introduced as a side effect of TF 2.2.4.

4. Erroneous record length for SYSIN causes a bomb instead of the message INPUT ERROR.

FR Status Listing

2.2.0002.10	U	mysterious overtime
2.2.0002.11	U	bad jump in CODE, no dump
2.2.0002.12	U	bad jump in CODE, no dump
2.2.0007.2	F5	
2.2.0015.NS1	F5	
2.2.0016.7283	S	accept more than one digit on -SPACE
2.2.0016.7284	N	INPUT does not behave exactly as in BTL SNOBOL4 (see separate section)
2.2.0016.82172	F5	
2.2.0017.115	N	memory overflow recovery error, see separate note
2.2.0017.117	D	dupl can bomb if given very large integer arguments
2.2.0017.118	D	continuation lines are not recognized if control cards (illegally) intervene before them. (i.e. no error message is given.)
2.2.0017.119	D	-SPACE never listed even if -PRINT is on
2.2.0017.120	F5	
2.2.0017.121	S	internal optimization suggestion
2.2.0017.122	F5	
2.2.0017.123	N	input question, see separate note
2.2.0017.124	F5	
2.2.0019/201	F4	
2.2.0022.1	F5	
2.2.0024.NS1	N	
2.2.0030.NS1	F4	
2.2.0030.NS2	F5	
2.2.0031.NS1	F5	
2.2.0038.NS1	E	the OS V21 problem again
2.2.0038.NS2	N	problem with JCL

FR Status Listing cont'd

2.2.0041.NS1	FS	
2.2.0042.NS1	E	the OS V21 problem again
2.2.0042.1	F5	
2.2.0056.1	F5	
2.2.0056.2	N	JCL problem

Please use FR forms wherever possible. Extra copies are enclosed with this issue of the newsletter.

Note on Input Compared with BTL SNOBOL4

There are some slight differences between SPITBOL and BTL SNOBOL4 which are illustrated by the following examples. These are not considered bugs and no attempt will be made to change SPITBOL to duplicate the apparently strange behavior of BTL SNOBOL4.

INPUT	This statement reads a line in SPITBOL, but not in BTL SNOBOL4.
A = INPUT EQ(1)	This statement reads a line in SPITBOL, but not in BTL SNOBOL4.

The one case in which SPITBOL fails to read a line when expected is a pattern match where the pattern evaluation fails and there is a replacement part.

INPUT EQ(1) =	Neither version reads a line for this statement.

Note on OS R21 Problem

OS V21 as distributed contains a serious error which disables many features in SPITBOL. This error must be corrected as outlined below. (This is a repeat of a previously issued memo.) It can be fixed by the following IBM APAR:

```
IBM OS APAR 53361
"SPIE EXIT REG14 INCORRECT"
OS Release 21

ZAP     NAME IEANUC01 IEAQNU00
        VER  00D8  58F0,0314
        REP  00D8  90EF,0314
```

Note "0314" should be "0214" at some installations.

Contact Local IBM Program Support Customer Engineer and give him the APAR number 53361 and symptom "SPIE EXIT REG14 INCORRECT" and he will check out fix for your installation.

The above problem caused a large number of ABENDS since register 14 was not restored after a program interrupt and that register is used as a SPITBOL base register.

Note on Conversion of Integers

The current version of SPITBOL (i.e. version 2) regards an all blank field as an error when converted to integer or real. This is a response to suggestions made some time ago. Unfortunately the manual was not changed and this accounts for the discrepancy in the manual. Leading and trailing blanks are still permitted as always.

Note on Error Recovery

The SETEXIT error recovery facility is not totally reliable in the case of memory overflow. Some operations, e.g. pattern matching,consume memory before checking overflow. Repeated application of such operations in the recovery routine will eventually cause an abort.

We are attempting to remove these situations, however in some cases the cost appears excessive. Meanwhile you recover from category 6 errors at your own risk!

New 360 Version of SPITBOL

A new source release of SPITBOL correcting all known errors and adding some new features is planned for the near future. A major component of this new release is a new system intefface for OS which will implement reading and writing of PDS members, multiple file tape handling, file tracing, and many other new features.

Other New Versions of SPITBOL

The DOS version is now available for distribution. The machine independant version is nearing completion. Initial versions deriving from this will be as follows:

1. PDP 11/45
2. ICL 1900
3. 360/67 TSS 32 bit addressing
4. CALL 360 version

The 1108 version is running locally and has successfully run many programs. We should be able to accept 1108 version orders in the near future.

Suggestions

Just one note this month. The following is a new feature which has been added to the machine independant version and will be retrofitted to all other versions. We call it the selection or alternative feature.

Anywhere that a value is expected, a series of expressions separated by commas, enclosed in parentheses can be written:

$(e_1, e_2, e_3 \ldots e_n)$

The semantics is to evaluate the expressions from left to right until one succeeds and then use its value. Failure is signalled if all evaluations fail.

This feature trivially provides an "or" function for predicates but also has many other uses as shown by the following examples:

```
A      = (EQ(B,3),GT(B,20)) B + 1

NEXT   = (INPUT,'%EOF')

MAXAB  = (GT(A,B) A,B)
```

The alternatives structure provides a limited IF-THEN-ELSE capability, and as such is a useful programming feature. It is our feeling that no additional attempt should be made to enlarge the conditional structure at this time. Note incidentally that the semantics of ordinary parentheses is a correct degenerate case of an alternative structure with one alternative.

Errors fixed by TF 2.2.6

1. Patterns with string elements longer than 256 characters fail to match correctly.

2. The compiler has an error preventing use of SPITBOL under VS-2. (Other operating systems are not affected.)

3. An error (introduced in TF 2.2.5) causes large programs to compile incorrectly. This accounts for all reported cases of programs running under 2.2.4 and failing under 2.2.5.

Note: Another error has been discovered but cannot be fixed till the next source release: keywords for pattern primitives (&ARB, &BAL, etc.) may not be used if object modules are to be generated.

SPITBOL NEWSLETTER

#5

July 1974

Robert B.K. Dewar
Kenneth Belcher
John Cole
Illinois Institute of Technology

Installations

There are now 101 SPITBOL installations, including ones in Canada, Brazil, Scotland, England, Germany, Holland, France, Israel, and Sweden.

SPITBOL/360 Plans

Version 3.0 has been delayed a long time, partly because of uncertainty over what our future directions should be and partly because of a temporary direction of effort to other work (see announcements later on in this section). We have now firmed up our plans as follows:

In October 1974 we will release the new interface which provides such features as reading and writing of PDS members, multiple tape file handling and file tracing. We will probably include a stability release of SPITBOL itself which will correct all known errors but contain no new features.

Meanwhile we are commencing work on a major rewrite, which will contain several new features, including syntactic extensions similar to those recently proposed by R. Griswold (SNOBOL-X, see SIGPLAN Notices, February 1974). Full details of these extensions will be published in a separate newsletter to be issued shortly.

We need your input now on directions in which we should move.

1108 Version of SPITBOL

A new version of SPITBOL for the UNIVAC 1106/1108/1110 machines is now available for distribution. Write to:

SPITBOL Project
Department of Computer Science
Illinois Institute of Technology
Chicago, Illinois 60616

for full details. This version is essentially upward compatible with the existing 360/370 version.

ICL 1900 Version of SPITBOL

The ICL 1900 version of SPITBOL, based on our machine independent version, is up and running. This version is being handled through Leeds University. Write to:

Dr. A.P. McCann
SPITBOL Project
Computational Sciences
University of Leeds
Leeds, Yorkshire,
ENGLAND

for details.

Macro-version of SPITBOL

An adaptation of the macro-version of SPITBOL for the PDP/11 is underway. We are happy to receive inquiries concerning implementations on other machines.

Techniques Department

A contribution from:

Dr. J.H. Lindsay
Department of Computing and Information Science
Queen's University
Kingston, Canada K7L 3N6

is attached.

Games?

We received an interesting contribution from Robert Hsu, Linguistics Department, University of Hawaii, Honolulu, Hawaii 96822. It is entitled "The String Game" and is too long to reproduce here, but copies can be obtained by writing Mr. Hsu directly.

CRUDE IMPLEMENTATION OF PSEUDO-VARIABLES IN SNOBOL.

```
      1           &TRACE = 3000
          *
          *   ESTABLISH THE PSEUDO-VARIABLE CREATION MECHANISM.
          *
      2           DEFINE('PSEUDO_VARIABLE(PVDEF,PVLABEL)PVN,PVAA,PVA,LHS2,LHS1,'
          +               'PVARGS')
          *
          *   SET UP SOME VALUES OF VARIABLES FOR TEST PURPOSES.
          *
      3           X = 'ORIGINAL VALUE OF X'
      4           Y = 'ORIGINAL VALUE OF Y'
      5           Z = 'ORIGINAL VALUE OF Z'
      6           A = 'ORIGINAL VALUE OF A'
      7           B = 'ORIGINAL VALUE OF B'
      8           C = 'ORIGINAL VALUE OF C'
          *
          *   SET UP THE PSEUDO-VARIABLE.
          *
      9           PSEUDO_VARIABLE('LHS(X,Y,Z)A,B,C')
          *
          *   USE IT.
          *
          *      VARIABLE WITH THE NAME OF THE PSEUDO-VARIABLE WILL HAVE THE VALUE
          *      ASSIGNED THE PSEUDO VARIABLE "FUNCTION"; THE ARGUMENTS OF THE
          *      PSEUDO-VARIABLE WILL BE EVALUATED AND SAVED BEFORE THE RHS OF THE
          *      ASSIGNMENT OR WHEN A PATTERN INVOLVING THE PSEUDO VARIABLE ON THE
          *      RIGHT OF THE BINARY . OR $ OPERATORS IS BEING FORMED.  THIS
          *      METHOD HAS A NUMBER OF UNDESIRABLE SIDE EFFECTS: MULTIPLE USES
          *      OF THE PSEUDO-VARIABLE IN THE SAME STATEMENT MAY CAUSE ANY
          *      EXCEPT THE LAST "EVALUATED" WITH DIFFERENT ARGUMENTS FROM THE
          *      LAST "EVALUATED" TO BE CALLED TO RECEIVE THE INPUT WITH THE
          *      ARGUMENTS OF THE LAST "EVALUATION".  SIMILAR CONSIDERATIONS
          *      APPLY TO THE USE OF THE PSEUDO VARIABLE IN ANY SORT OF RECURSION.
          *      IT IS MEANINGLESS FOR A TRACE ROUTINE OR A PSEUDO-VARIABLE
          *      ROUTINE TO FAIL OR NRETURN.
          *
          *      SINCE THE MECHANISM DEPENDS ON TRACING, &TRACE MUST BE SET HIGH
          *      ENOUGH TO ALLOW FOR ALL CALLS TO PSEUDO-VARIABLES DURING
          *      EXECUTION OR PSEUDO-VARIABLE ROUTINES WILL NOT BE CALLED AFTER
          *      &TRACE HAS BEEN DECREMENTED TO ZERO.
          *
          *      THERE IS TOO MUCH OVERHEAD IN ALL THIS; IT COULD BE DONE MUCH
          *      MORE EFFICIENTLY AND WITHOUT THE GLITCHES INTERNALLY SINCE THE
          *      TRACED VARIABLE WHICH IS NRETURNED COULD BE ASSOCIATED WITH THE
          *      PARTICULAR CALL.
          *
     10           LHS('TEST-X','TEST-Y','TEST-Z') = 'SOME JUNK'    :(END)
          *
          *   PSEUDO-VARIABLE ROUTINE TO BE CODED BY THE USER.
          *
     11 LHS       OUTPUT = LHS '|' X '|' Y '|' Z '|' A '|' B '|' C '|' :(RETURN)
```

CRUDE IMPLEMENTATION OF PSEUDO-VARIABLES IN SNOBOL.

```
      *    THIS IS THE CODE TO CREATE THE PSEUDO-VARIABLE.
      *
   12 PSEUDO_VARIABLE
      *
      *       PSEUDO_VARIABLE FIRST BREAKS ITS FIRST ARGUMENT UP INTO A
      *       FUNCTION NAME PVN, AN ARGUMENT STRING PVAA, AND A TEMPORARY
      *       VARIABLE STRING PVDEF.  IT THE GRINDS THROUGH THE ARGUMENT
      *       STRING (E.G. 'X,Y,Z') TO PRODUCE A STRING OF SNOBOL STATEMENTS IN
      *       LHS2 TO SAVE THE ARGUMENT VALUES X, Y, Z IN VARIABLES
      *       X__1, Y__1, Z__1 AND ANOTHER STRING OF SNOBOL STATEMENTS TO
      *       RESTORE THE VALUES FROM X__1, Y__1, ETC. TO X, Y, ETC.  IF A
      *       SECOND ARGUMENT IS NOT GIVEN, PSEUDO_VARIABLE USES THE PSEUDO-
      *       VARIABLE NAME AS THE ENTRY POINT OF THE USER'S PSEUDO-VARIABLE
      *       CODE.  PSEUDO_VARIABLE THEN COMPILES A ROUTINE (E.G. LHS(X,Y,Z))
      *       WHICH WILL SAVE THE VALUES OF THE ARGUMENTS IN HIDDEN VARIABLES
      *       AND NRETURN A TRACED VARIABLE (E.G. LHS__1).  PSEUDO_VARIABLE
      *       ALSO COMPILES A PROLOGUE TO THE USER'S PSEUDO-VARIABLE ROUTINE
      *       (E.G. LHS__2(LHS__VARNAME,LHS__TAG)LHS,X,Y,Z,A,B,C) TO PUT THE
      *       VALUE ASSIGNED TO THE PSEUDO-VARIABLE INTO THE VARIABLE WITH THE
      *       SAME NAME AS THE PSEUDO-VARIABLE ROUTINE, PLACE THE SAVED
      *       VALUES OF THE ARGUMENTS IN THE ARGUMENT VARIABLES , AND GO TO
      *       THE USER'S PSEUDO-VARIABLE ROUTINE.  THIS PROLOGUE IS MADE THE
      *       TRACING ROUTINE FOR THE TRACED VARIABLE NRETURNED BY THE FIRST
      *       ROUTINE COMPILED AS MENTIONED ABOVE.
      *
   13           PVDEF BREAK('(') . PVN '(' BREAK(')') . PVAA ')' = :F(FRETURN)
   14           PVARGS = PVAA ','
   15 PV__A     PVARGS BREAK(',') . PVA ',' = :F(PV__C)
   16           LHS2 = LHS2 ' ' PVA '__1 = ' PVA ';'
   17           LHS1 = LHS1 ' ' PVA ' = ' PVA '__1;' :(PV__A)
   18 PV__C     PVLABEL = IDENT(PVLABEL) PVN
   19           CODE(PVN '__1' LHS2 ' ' PVN ' = .' PVN '__1 :(NRETURN)')
      +                :F(FRETURN)
   20           DEFINE(PVN '(' PVAA ')',PVN '__1') :F(FRETURN)
   21           CODE(PVN '__2' LHS1 ' ' PVN ' = $' PVN '__VARNAME :('
      +              PVLABEL ')') :F(FRETURN)
   22           LHS1 = PVN '__2(' PVN '__VARNAME,' PVN '__TAG)' PVN
   23           LHS1 = DIFFER(PVAA) LHS1 ',' PVAA
   24           LHS1 = DIFFER(PVDEF) LHS1 ',' PVDEF
   25           DEFINE(LHS1) :F(FRETURN)
   26           TRACE(PVN '__1',,,PVN '__2') :F(FRETURN)S(RETURN)
      *
   27 END
```

SOME JUNK|TEST-X|TEST-Y|TEST-Z||||