BASIC

SPECIFICATIONS

technical report
73A-1.1

A Report Prepared by

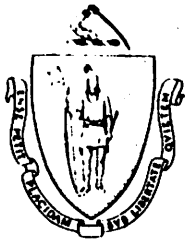
John A. N. Lee, Professor and Director
Steven R. Beckhardt, Research Assistant
Arthur I. Karshmer, Research Assistant

Computer and Information Science Program
University of Massachusetts
Amherst, Mass., 01002

On Behalf Of

U.S. Department of Commerce
National Bureau of Standards
Washington, D.C. 20234

July, 1973



The Commonwealth of Massachusetts

University of Massachusetts

Amherst 01002

1973 August 01

This report forms one part of a document set which was developed under contract to the National Bureau of Standards. This report and its companion parts will be considered for Federal standardization and is intended also to assist in the national standardization effort under the auspices of American National Standards committee X3J2.

The complete set of documents are available from the National Technical Information Service as report NBS-GRC 73-17.

The parts are:

Part 1 : Standard Specifications for BASIC

Part 2 : Explanations, Examples and Recommendations

Part 3 : Programs for the Validation of BASIC Implementations

Part 4 : Formal Description of BASIC

1. INTRODUCTION

1.1 Purpose

This specification establishes the form for and the interpretation of programs which are written in the BASIC language for the purpose of promoting a high degree of interchangeability and transportability of such programs for use in a variety of automatic data processing systems. An implementation shall conform to this specification provided that it accepts, and interprets as herein specified, at least those forms and relationships described herein.

Whereas the original implementations of this language were intended for use in automatic data processing system which supported some degree of interaction between the system and the user, the use of these specifications is not so restricted.

1.2 Scope

This specification establishes:

(1) The form of a program written in the BASIC language.

(2) The form in which data shall be written for input into a automatic data processing system being controlled by a program written in the BASIC language.

(3) The semantic rules for interpreting the meaning of a program.

(4) The form in which an automatic data processing system being controlled by a program written in the BASIC language shall output numeric and string data.

This specification does not prescribe:

(1) The mechanism by which programs written in the BASIC language are transformed for use by an automatic data processing system.

(2) The method of transcription of programs written in the BASIC language, or their input or output data, to or from a data processing medium.

(3) The means by which programs written in the BASIC langauge are executed in the control of a supervisory system.

(4) The minimum requirements of an automatic data processing system which is capable of supporting an implementation of a processor for the BASIC language.

(5) The bounds which may be placed on the size of a program written in the BASIC language, or any of the elements of those programs. This specification does, however, place certain minimum requirements on certain program elements.

2. TERMINOLOGY

Where a term is used in this specification whose meaning is not described in this section, the interpretaion given by American National Standard X3.12-1968 shall be applicable.

overflow
A condition which is said to exist in an automatic data processing system when a prior arithmetic operation has resulted in a representation whose magnitude is greater than that of the greatest value that can be represented in the intended unit of storage.

underflow
A condition which is said to exist in an automatic data processing system when a prior arithmetic operation has resulted in a nonzero result whose magnitude is less than that of the smallest nonzero value that can be represented in the intended unit of storage.

error
Within the domain of this standard the term error is to be taken to mean that the implementation has recognized that the program is not performing within the specifications herein. For the purposes of conciseness, the specifications do not include specific reference to syntactic errors in the text of the program. It is to be assumed that where a statement or program element does not conform with the syntactic rules given, an error condition automatically exists.

Error conditions are classified in two manners:

fatal error
an error of such magnitude that the execution of the program is immediately suspended. If the error is determined during the compilation phase of the implementation, if any, the compilation may continue, but the execution of the program may not be initiated.

non-fatal error
an error of less magnitude than a fatal error, wherein the user is informed of the error condition but the execution of the program is allowed to continue, the implementation having taken specified corrective measures.

Such corrective measures may include a request from the implementation to the user to provide the correction.

3. THE LANGUAGE ELEMENTS

3.1 The character sets

A program is written using the character set which is prescribed by the ASCII character set (American National Standard Code for Information Interchange, X3.4-1968).

The alphabetic character set is the set of roman letters which are contained in ASCII in positions 4/1 through 5/10 or positions 6/1 through 7/10, which correspond to the upper and lower case character sets respectively.

The set of decimal digits is the set of arabic digits contained in ASCII in positions 3/0 through 3/9.

The set of ASCII characters which are termed Graphic Characters in American National Standard X3.4-1968, are acceptable to the language, though only a specific subset have specific meanings within the text of the programs. This set of graphic characters is termed the set of special characters herein.

3.2 Lines of Program Text

A line of text in a program written in the BASIC language corresponds to a statement. A line is a string of characters drawn from the character sets specified in section 3.1. Each line is composed of two parts:

- a line number (see section 4.3) and
- a program statement.

The line number assigned to each line must be unique. The length of each line of program text is limited by the implementation. A program statement must be contained totally within one line of text; no provisions are made within this specification for the continuation of statements over more than one line.

Similarly no provision is made for the placement of several statement on one line. Spaces contained in a line of text are not significant except where they are contained within string constants.

3.3 General Statement Form

A program statement consists of two components:

- a keyword which specifies the class of action specified by the statement, and
- a set of arguments necessary to the action expected, syntactically sugared to provide a more readable program.

4. DATA ELEMENTS

4.1 Numeric Constants

General Syntactic Form

A numeric constant is a decimal representation in positional notation of a number. The value of a numeric constant cannot be altered during the execution of the program. There are four general forms of a numeric constant:

(NR1 or implicit point representation) d...d
(NR2 or explicit point unscaled representation) d...drd...d
(NR3 or explicit point scaled representation)
d...drd...dEsd...d
(implicit point scaled representation)
d...dEsd...d

where

- d is a decimal digit,
- r is a decimal point,
- s is an optional sign and
- E is the explicit character E.

The numeric constants accepted by this language include as a subset, the representations specified by the proposed American National Standard X3.42-1973, "The Representation of Numeric Values in Character Strings for Information Interchange".

Description

Within the text of the program, with the exception of numeric value representations in DATA statements (see section 7.2.1.2) and PRINT statements, where the numeric representation is not an operand in an expresssion (see section 7.1.3.1), all numeric representations are unsigned. The set of numeric representations accepted by the implementation may be restricted provided that the following minimum requirements are satisfied:

a) In all numeric representations, the implementation shall accept at least six (6) significant digits of precision,
b) In the explicit-point-scaled representation (NR3 form), the implementation shall accept representations which include at least two significant digits in the exrad component.

Exceptional Cases

Numeric representations expressed in the explicit-point-scaled representations which do not include an explicit sign in the exrad component shall be permitted; the sign of the exrad component is to be interpreted as being positive in this case.

Recognizable Error Conditions

Since the implementation is permitted to restrict the set of numeric representations which are acceptable, error conditions may arise when the numeric value represented is outside the acceptable range.

4.2 String Constants

General Syntactic Form

A string constant is a character string enclosed in quotation marks.

Description

The value of a string constant cannot be altered during the executio of the program.

The length of a string constant, as defined by the number of characters contained in its character string, shall be in the range from zero (0) characters to an implementation defined maximum which shall not be less than six (6) characters. The number of characters in a string constant is fixed during the execution of the program.

Exceptional Cases

The string of length zero (0) is known as the null or empty strin and is represented as a string constant by contiguous quotation marks. That is, "".

To embed a quotation mark within a string constant, two contiguous quotation marks are used. This is distinguishable from the null string by the context of the surrounding string and its enclosing quotation marks. That is, the string constant "A""B" represents the character string A"B .

String constants which are contained in DATA statements or in an input line which is the response to the request for data from an INPUT statement, need not be contained in quotation marks. However, in this context, the delimiting character of a string is the comma and hence commas cannot be embedded in these strings. Similarly, the null string cannot be represented without quotation marks in DATA statements or in data input from the users terminal.

Recognizable Error Conditions

Where the length of the string constant is in excess of that accepte by the implementation a fatal error message shall be output.

4.3 Line Numbers

General Syntactic Form

n

where:

n is one (1) to five (5) decimal digits with no embedded space characters.

Description

Line numbers are used to label and reference statements. Every statement must be preceded by a line number. Two line numbers which differ syntactically are equivalent if they represent the same numeric value.

Exceptional Cases

If the line number of a non-executable statement is the object of a transfer of control statement (see section 7.1.2), execution continues at the first executable statement physically sequential to the non-executable statement.

Recognizable error conditions

Reference to a non-existent line number shall result in a fatal error message.

5. VARIABLE ELEMENTS

A variable is a datum that is identified by a symbolic name. Such a datum may be referenced and defined in the program. In this standard two classes of datum are recognized:
numeric datum and string datum;
correspondingly there exist two classes of variables which are associated with these data types, called herein
numeric variables and string variables.
Each variable type is identified syntactically by distinctive symbolic names; see sections 5.1 and 5.2 respectively.

5.1 Numeric Variables

Numeric Variables identify numeric data elements.
There exist two classes of numeric variables:
a) scalar numeric variables which are associated at any instant in the execution of the program

with a scalar numeric value, and
b) numeric array variables which identify an ordered set of numeric values.

5.1.1 Scalar Numeric Variables

General Syntactic Form

A scalar numeric variable is named by an alphabetic character followed by an optional digit.

Description

A scalar numeric variable can be associated only with a scalar numeric value. The value initially associated with a scalar numeric variable, prior to the first reference to that variable in an executable statement, shall be zero (0).

Exceptional Cases

The name associated with a numeric scalar variable can also be used as the name of an numeric array variable; however, since a numeric array variable is always syntactically associated with a parenthesized set of indices, there is a clear distinction between the two usages of the same name.

Recognizable Error Conditions

None

5.1.2 Numeric Array Variables

General Syntactic Form

A numeric array variable is named by a single alphabetic character.
A reference to an element of a numeric array has the form:

a(e-1) or a(e-1 , e-2)

where
a is a numeric array name and
the e-i are arithmetic expressions.

Description

A numeric array is an ordered set of scalar numeric values. In this language arrays may be either one-dimensional or two-dimensional. Reference to an element of an array requires the identification of the element by the provision of the name of the array in which it is contained and the indices of its

position within the array. The indices are expressed in terms of arithmetic expressions and are called subscripts. Where the array is one-dimensional, one subscript expression must be provided with each reference; where the array is two-dimensional, two subscript expressions must be provided with each reference.

The subscript expressions are evaluated (see section 6.1) prior to each reference to an array element; if the value of any subscript expression is not an integer, then the value is truncated to the value of the greatest integer which is less than or equal to the value of the subscript expression. That is,

is-integer(r) and $s-1 < r \leq s$

where s is the value of the subscript expression and r is the resulting index value.

The number of dimensions and the dimensions of each numeric array are specified either
a) by their explicit specification in a DIM statement (see section 7.2.1.1), or
b) by the occurrence of a numeric array element in an executable statement.

In the latter case, the implicit upper bound on each dimension shall be ten (10).
The lower bound on the dimension of a numeric array is zero (0).
The indices of each dimension of a numeric array shall be continuous from the lower bound to the upper bound.
The value initially associated with each element of a numeric array, prior to the first reference to any element of the array in an executable statement, shall be zero (0).

Exceptional Cases

The name associated with a numeric array variable can also be used as the name of a scalar numeric variable; however, since a numeric array variable name is always syntactically associated with a parenthetical set of indices, there is a clear distinction between the two usages of the same name.

Recognizable Error Conditions

A fatal error message shall be output when:
a) The number of subscript expressions in a numeric array element reference does not equal the number of dimensions specified either explicitly in a DIM statement, or implicitly by a previous reference to an element of that same numeric array variable.
b) The evaluated subscript expression specifies an element of the numeric array which is outside the bounds specified by the dimensions; that is, greater than the upper bound or less than the lower bound.
c) The array name occurs in more than once in DIM statements.

5.2 String Variables

General Syntactic Form

A string variable is named by an alphabetic character followed by a decimal digit and a dollar sign ($) or an alphabetic character followed by a dollar sign ($).

Description

A string variable identifies a character string datum. The length of any character string datum associated with a string variable can vary during the execution of the program, from a length of zero (0) characters (signifying the null or empty string) to an implementation defined maximum which shall not be less than six (6) characters.

The initial value of the character string associated with a string variable, prior to the first reference to that string variable in an executable statement, shall be the null (or empty) string.

Exceptional Cases

None

Recognizable Error Conditions

None

5.3 Functions

General Syntactic Form

A function name is a three (3) character string; the names of implementation supplied functions (see section 8.2) are agreed between the implementation and the user. Names of user defined functions (see section 7.2.1.3 for definition) have the form

FNa

where a is an alphabetic character.

Description

Function names are used to identify predefined algorithms which evaluate arithmetic functions.

Exceptional Cases

None

Recognizable Error Conditions

see section 8.

5.3.2 Function References
(see section 8.1)

5.3.3 Parameter Names

General Syntactic Form

ad

or

a

where

a is an alphabetic character and
d is a decimal digit.

Description

Parameters are named by the same syntactic construct as scalar numeric variables are named, since their usage in DEF statements (see section 7.2.1.3) is identical to the usage of scalar numeric variables in arithmetic expressions. However, the scope of parameters is restricted to the DEF statement in which they are specified in the parenthesized list to the left of the equals (=) symbol. On evaluation of the function in which they appear, parameters are associated on a one to one basis with the evaluated arguments which are specified in the current function reference. The mode of parameters is the same as the mode of their corresponding arguments; that is, numeric.
No initial values are associated with parameters.

Exceptional Cases

The names assigned to parameters can also be used as the names of scalar numeric variables in the program outside of the DEF statement in which the parameters are specified. The same names can be assigned to parameters in other DEF statements, the scope of each parameter (specified above) unambiguously distinguishing the applicability of the name to a datum.
Reference to a name within the expression which defines a function in a DEF statement is a reference to a scalar numeric variable in the set of executable statements in the body of the program, provided that there exists no parameter of the same name in the DEF statement.

Recognizable Error Conditions

None

6. EXPRESSIONS

Expressions, which when evaluated resulted in a value, are either numeric or string expressions. There does not exist in this language any form of "mixed mode" expression. String expressions are restricted to be either scalar string variable or string constants.

6.1 Numeric Expressions

General Syntactic Form

op-1 t-1 op-2 t-2 op-3 t-3 ... op-n t-n

where:

each operator op-i (for i > 1) is one of the following character
1) + the operator which represents addition,
2) - the operator which represents subtraction,
3) * the operator which represents multiplication,
4) / the operator which represents division, or
5) ↑ or ** the operator which represents involution.

and where
the operator op-1 if present, is one of the following characters
1) + the operator which represents unary addition, or
2) - the operator which represents negation.

The operator op-1 is optional.

Each term t-i shall be one of the following
1) a numeric constant
2) a numeric variable (scalar variable or array element)
3) a function reference
4) a numeric expression enclosed by parentheses

There shall exist at least one term t-i in each numeric expression and one operator op-i (i > 1) for each additional term t-i .

Description

Each numeric expression represents a sequence of numeric operations to be performed over the given set of numeric constants and numeric variables. Numeric

expressions are composed of operators, numeric constants, numeric variables, function references, and parentheses according to the following rules:

1) A numeric expression is composed of an optional prefix (unary) operator followed by a multiply factor
or
an expression followed by an addition or subtraction operator followed by a multiply factor.

2) A multiply factor is composed of an involution factor
or
a multiply factor followed by a multiplication or division operator, followed by an involution factor.

3) An involution factor is composed of a term
or
a term followed by an involution operator followed by another term.

4) A term is as defined above.

Evaluation of Expressions

A part of an expression need be evaluated only if such action is necessary to establish the value of the expression. The syntactic rules for the formation of expressions imply the binding strength of operators. The evaluation of an expression may proceed according to any valid formation sequence.
When two operands are to be combined by an operator, the order of evaluation of the individual operands is optional. If the mathematical use of operators is associative, commutative or both, full use of these properties may be made in order to revise the orders of combination, provided only that the integrity of parenthesized expressions is not violated.
Reference to an element of an array requires the evaluation of its subscript expression(s) prior to the usage of the reference in its embedding expression.

Exceptional Cases
Where the expression is part of a DEF statement (see sec. 7.2.1.3) the set of permitted terms is extended to include the set of parameters defined in the DEF statement, and is restricted to disallow any references, either direct or indirect, to the function being defined.

In each and every evaluation of an expression which defines a user defined function, parameters are replaced by the value of the

corresponding argument in the current function reference.

Recognizable Error Conditions

1) A reference to an undefined function shall yield a fatal error.

2) If, during the evaluation of an expression, a number is obtained which is outside the range of the permitted values of the implementation, the following action shall be taken:

   a) An appropriate error message shall be generated.
   b) On overflow, the magnitude of the result shall be replaced by the maximum magnitude representable by the implementation.
      On underflow, the result shall be replaced by zero.
      On an attempted division by zero, the division operation shall be abandoned and the result set equal to the magnitude of the greatest representable value in the implementation, with the sign of the dividend.
   c) Execution shall be continued.

3) Zero raised to a negative or zero power shall result in a fatal error.

4) A negative number raised to a non-integral power shall result in a fatal error.

5) For subscript errors see section 5.1.2; Numeric Array Variables.

6.2 String Expressions

General Syntactic Form

string constant (see section 4.2)
or
string variable (see section 5.2)

Description

A string expression represents a string value which is to be utilized in the execution of other statements. The value of a string constant is the string which it represents. The value associated with a string variable is the string which was last assigned to it in an assignment statement, a READ statement or an INPUT statement.

Exceptional Cases

None

Recognizable Error Conditions

None

6.3 Relational Expressions

General Syntactic Form

e-1 r-op e-2

where e-i are expressions
and r-op is a relational operator (see sec. 6.2.1)

Description

The evaluation of a relational expression results in the determination of a truth value, that is, TRUE or FALSE. The truth value of a relational expression is determined by the truth or falsity of the relation expression.

The mode of the two expressions, e-1 and e-2, shall be equal; that is, if the mode of expression e-1 is numeric then the mode of expression e-2 must also be numeric. Where the mode of expression e-1 is a string, then the mode of expression e-2 must also be a string.

Arithmetic expressions are described in section 6.1.

String expressions are restricted to either a string constant or a string variable. Where the expressions are of mode string, the only defined relational operators are equals (=) and not equals (<>).

Two strings (identified either by a string variable or as a character string enclosed in quotation marks) are equal if and only if:

   a) the strings are both of the same length and,
   b) both strings totally contain the exact same sequence of characters.

Exceptional Cases

None

Recognizable Error Conditions

A fatal error message shall be output if the mode of expression e-1 is not identical to the mode of expression e-2.

6.3.1 Relational Operators

General Syntactic Form

=	(equals)
<>	(not equals)
<	(less than)
<=	(less than or equal to)
>	(greater than)
>=	(greater than or equal to)

Description

Relational operators which express the relationships of equality (=) and inequality (<>) are defined over the domains of both numeric values and strings. The relational operators, less than (<), less than or equal to (<=), greater than (>) and greater than or equal to (>=), are defined only in the domain of numeric values.

Where two strings are being related in a relational expression (see sec. 5.2), the two strings are equal if and only if:

a) the strings are both of the same length and,
b) both strings totally contain the exact same sequence of characters.

The relationships between numeric values correspond to the normal mathematical relationships over the domain of real numbers.

Exceptional Cases

None

Recognizable Error Conditions

If any of the relational operators that are defined only in the domain of numeric values are used to test the relationship of string constants or string variables, a fatal error message shall be output.

7. PROGRAM STATEMENTS

7.1 Executable Statements

7.1.1 Assignment Statements

7.1.1.1 Numeric LET statements

General Syntactic Form

LET v-1, v-2, v-3, ... ,v-n = e

where:

each v-i is a numeric variable (scalar variable or array element), and e is an arithmetic expression.

Notes: The word LET is optional. There must be at least one numeric variable v-i.

Description

The arithmetic expression is evaluated and the result is assigned to the numeric variables. If any of the numeric variables are array elements, the subscript expressions are evaluated prior to assignment.

Exceptional Cases

None.

Recognizable Error Conditions

1) If the expression, e, is a string expression, a fatal error shall result.

2) For subscript errors, see section 5.1.2; Numeric Array Varaiables.

3) For arithmetic expression errors, see section 6.1; Arithmetic Expressions.

7.1.1.2 String LET statements

General Syntactic Form

LET s-1, s-2, s-3, ... , s-n = se

where:

each s-i is a string variable,
and se is a string expression (string variable or string constant).

Notes: The word LET is optional. There must be at least one string variable s-i.

Description

The string expression is evaluated and the result is assigned to the string variables.

Exceptional Cases

None.

Recognizable Error Conditions

If the expression, se, is an arithmetic expression, a fatal error shall result.

7.1.2 Control Statements

7.1.2.1 GO TO Statement

General Syntactic Form

GO TO n
or
GOTO n
where
n is a line number (see section 4.3).

Description

The execution of a GO TO statement causes the normal sequence of execution of statements to be altered by specifying that the next statement to be executed is that which is labelled with the line number n. The normal sequence of execution is to be followed thereafter, or until another transfer of control statement is executed.
See also section 4.3, line numbers.

Exceptional Cases

If the line number of a non-executable statement, other than the END statement of the program, is the object of the GO TO statement, then the new sequence of execution is to continue at the first executable statement, which is physically sequential to the non-executable statement.

Recognizable Error Conditions

If the object of the GO TO statement is a non-existent statement, a fatal error message shall be output.

Transfer of control into the range of a FOR-NEXT block shall be restricted unless prior execution processes have been followed, see section 7.1.2.3.

7.1.2.2 IF Statement

General Syntactic Form

IF r-e THEN n

where r-e is a relational expression
(see sec. 6.2, RELATIONAL EXPRESSIONS)
and n is a line number

Description

If, after evaluation, the relational expression, r-e, is true, program control is transferred to the statement labelled with the line number specified by n. If the relation is false, program execution continues at the next physically sequential executable statement after the IF statement.

Exceptional Cases.

If the line number of a non-executable statement, other than an END statement of the program, is the object of the IF statement, then the new sequence of execution is to continue at the first executable statement which is physically sequential to the non-executable statement.

Recognizable Error Conditions.

If during the execution of an IF statement, an attempt is made to transfer program control to a non-existent line number, n, a fatal error message shall be output and execution of the program shall be halted.

Transfer of control into the range of a FOR-NEXT block shall be restricted unless prior execution processes have been followed,

see section 7.1.2.3.
7.1.2.3 FOR-NEXT blocks

General Syntactic Form

FOR statement
block
NEXT statement

where:

the FOR statement is described in section 7.1.2.3.1, FOR statements
the NEXT statement is described in section 7.1.2.3.2, NEXT
statements.
and block is a collection of statements.

Description

The statements in the FOR-NEXT block are executed repeatedly according to the specifications in the FOR statement. The scalar numeric variable in the NEXT statement must be the same variable as the control variable in the FOR statement (see sections 7.1.2.3.1 FOR statements and 7.1.2.3.2 NEXT statement). This variable is known as the control variable of the FOR-NEXT block. A FOR-NEXT block may contain other FOR-NEXT blocks as long as the contained blocks are subsets of the containing block. These are said to be nested FOR-NEXT blocks. Control can only enter a FOR-NEXT block through the FOR statement unless the FOR-NEXT block had already been entered and then exited before the FOR statement was satisfied (see section 7.1.2.3.1 FOR statements). The control variables of two FOR-NEXT blocks cannot be the same if one block is contained within the other. The maximum depth to which FOR-NEXT blocks can be nested is implementation dependent, but shall not be less than four (4).

Exceptional Cases

None

Recognizable Error Conditions

1) A FOR-NEXT block that is only partially contained within another FOR-NEXT block shall be a fatal error.

2) Nested FOR-NEXT blocks with the same control variable

shall be a fatal error.

7.1.2.3.1 FOR statements

General Syntactic Form

FOR v = exp-1 TO exp-2 STEP exp-3

or

FOR v = exp-1 TO exp-2

where:

v is a scalar numeric variable called the control variable
exp-1 is an expression called the initial value
exp-2 is an expression called the limit
and exp-3 is an expression called the increment.

Description

A FOR statement (in conjunction with the NEXT statement) causes repeated execution of the statements in the FOR-NEXT block according to the following rules:

1) The expressions representing the initial value, limit, and increment are evaluated and their values are assigned to three uniquely named implementation supplied storage units herein called own1, own2, and own3 respectively. If the increment is not specified, it is assumed to be positive one (+1). The value of own3 cannot be zero (0).

2) If the value of the expression

(own1 - own2) x own3

is greater than zero (0) control is transferred to the statement following the corresponding NEXT statement. In this case, the FOR statement is said to be satisfied. Otherwise, the following steps are followed.

3) The control variable is set equal to the current value assigned to own1.

4) The statements in the FOR-NEXT block are executed.

5) When the corresponding NEXT statement is executed, the value of own1 is set equal to the sum of the values of the control variable and own3.

6) Step 2 is then repeated.

Notes: 1) The expressions representing the initial value, limit, and increment are evaluated only once, prior to the execution of the statements in the FOR-NEXT block.

2) After leaving a FOR-NEXT block, the value of the control variable is the last value it was assigned during execution of the FOR-NEXT block.

Exceptional Cases

None.

Recognizable Error Conditions.

1) A FOR statement without a matching NEXT statement shall be a fatal error.

2) A NEXT statement without a matching FOR statement shall be a fatal error.

3) A FOR statement whose increment (or own3) evaluates to zero (0) shall be a fatal error.

7.1.2.3.2 NEXT statements

General Syntactic Form

NEXT v

where v is a scalar numeric variable.

Description

The NEXT statement denotes the end of a FOR-NEXT block. The matching FOR statement is the closest, physically preceding FOR statement whose control variable is the same as the scalar numeric variable, v, in the NEXT statement. This shall always

by the closest, physically preceding FOR statement that has not already been matched with a NEXT statement. The execution of a NEXT statement is described in section 7.1.2.3.1 (FOR statements).

Exceptional Cases

None.

Recognizable Error Conditions

If the matching FOR statement is not the closest, physically preceding FOR statement that has not already been matched with a NEXT statement, then a fatal error message shall be output.

7.1.2.4 GOSUB-RETURN statements

The pair of statements GOSUB and RETURN operate together, the execution of the RETURN statement depending on the previous execution of a GOSUB statement.

7.1.2.4.1 GOSUB statements

General Syntactic Form

GOSUB n
where
n is a line number (see section 4.3).

Description

The execution of a GOSUB statement depends on the existance of a stack which operates on the last-in, first-out principle, which contains a set of line numbers. Initially, that is prior to the execution of the first GOSUB statement in the program, the line number stack is empty. On execution of a GOSUB statement, the line number of the physically next statement is placed on the line number stack and the normal sequence of execution of statements is broken, a new sequence being started at the statement which is labelled by the line number in the GOSUB statement.

When a RETURN statement is executed (see section 7.1.2.4.2), the normal sequence of executing statements is broken and a new sequence is started at the statement which is labelled by the line number which is at the top of the stack. That is, the

last line number placed on the stack by the execution of a GOSUB statement. This line number is removed from the stack on execution of the RETURN statement.

The execution of subsequent GOSUB statements without the execution of intervening RETURN statements is permitted; however the implementation may limit the size of the line number stack and hence the number of GOSUB statements that can be executed prior to the execution of a RETURN statement.

Exceptional Cases

Where the line number in the GOSUB statement references a declarative statement (see section 7.3) the new sequence of execution will commence with the first executable statement which physically follows the referenced statement.

Recognizable Error Conditions

Where the line number, n, references a non-existent statement a fatal error message shall be output.

(see also section 4.3, Line Numbers)

7.1.2.4.2 RETURN STATEMENTS

General Syntactic Form

RETURN

Description

The execution of a RETURN statement depends on the existance of a stack which operates on the last-in, first-out principle, which contains line numbers. The execution of the RETURN statement breaks the normal sequence of statement execution and a new sequence is commenced at the statement which is labelled with the line number which is on the top of the line number stack. This line number is then removed from the top of the stack.

Note- It is not necessary that on final completion of the execution of the program that the line number stack be empty.

Exceptional Cases

None

Recognizable Error Conditions

There can never exist the situation in which the line number which is on the top of the stack refers to a non-existent statement; the description of the GOSUB statement requires that the execution of that statement place on the stack the line number of the physically next statement.

When the line number stack is empty, the attempted execution of a RETURN statement shall cause a fatal error indicating that a RETURN statement execution is being attempted for which there was not a corresponding GOSUB statement.

7.1.2.5 ON statements

General Syntactic Form.

ON e GO TO line-1,line-2,line-3,...,line-n
or
ON e GOTO line-1,line-2,line-3,...,line-n

where e is any valid arithmetic expression
and each line-i is a line number.

Description.

When the ON statement is executed, the expression, e, is evaluated, and if necessary, truncated to an integer. If the resulting value is 1, program control is transferred to the statement labelled with the line number specified by line-1. If the resulting value is 2, control is transferred to the statement which is labelled with the line number specified by line-2, and so on.

(see also sec. 7.1.2.1, the GO TO statement)

Exceptional Cases.

If the line number of a non-executable statement, other than the END statement of the program, is the object of the ON statement, then the new sequence of execution is to continue at the first executable statement, which is physically sequential to the non-executable statement.

Recognizable Error Conditions.

If the value of the expression, e, is less than 1, or greater than the total number of line numbers, a non-fatal error message shall be issued, and execution is to continue at the next executable statement which is

physically sequential to the offending ON statement.

If during execution of an ON statement, an attempt is made to transfer control to a non-existent line number, a fatal error message shall be output.

Transfer of control into the range of a FOR-NEXT block shall be restricted unless prior execution processes have been followed, see section 7.1.2.3.

7.1.2.6 STOP statements

General Syntactic Form

STOP

Description

The execution of a STOP statement causes the execution of the program to be halted. The execution of the END statement has the same effect. (see also sec. 7.2.1.4)

Exceptional Cases

None

Recognizable Error Conditions

None

7.1.3 Input/Output statements

7.1.3.1 PRINT statements
General syntactic form

PRINT p-1 e-1 p-2 e-2 p-3 e-3 ... p-n e-n p-end
where
each print element e-i, is an expression, string variable, string constant, or null, and each element p-i, is a comma or semicolon.

Description

Execution of the PRINT statement causes each print element e-i, to be output according to a specified output format, with the initial character position being determined by the punctuation p-i.

The print line is to be divided into a fixed number of printing zones where the number of zones and the number of

character positions contained in each zone is implementation dependent, but constant from line to line. However the number of character positions contained in a zone shall not be less than that required to represent a number in explicit point scaled representation (American National Standard X3.42-1973, NR3 representation) with at least six significant digits. The punctuation p-i, controls the usage of the printing zones as follows:

1) A comma causes printing of the next print element to begin in the first (leftmost) character position of the next printing zone. If there are no more printing zones the current line, the ASCII characters carriage return and line feed are output and printing continues in the first (leftmost) printing zone of the next line. Multiple, contiguous commas may be used to specify spacing over several printing zones.

2) A semicolon causes printing of the next print element to begin at the next character position, with the following exception. If the print elements directly preceding and directly following the semicolon are numeric expressions, one space character is inserted between them on output.

3) When the last print element has been printed, a carriage return and a line feed are issued unless the last print element is followed by punctuation. In this case a carriage return and line feed are not output and subsequent printing continues on the same line according to 1 or 2 above as appropriate.

Output Format

1) string constants - the character strings which string constants represent (see section 4.2) are output in their entirety. Where a string cannot be output completely on the current line, the string is broken and the remainder is output starting in the leftmost position on the succeeding line.

2) string variables - The value associated with a string variable is printed in the same way as a string constant.

3) numeric expressions - Each numeric expression is evaluated and the result is printed conforming with the specifications of American National Standard X3.42-1973 in one of the following forms: signed NR1, signed NR2, or NR3 as specified below.

a) Each number is to be represented with not less than six digits of precision, unless that number can be represented exactly with less than six digits.

b) Each number that can be represented exactly as an integer is to be output in signed NR1 form with the exception of numbers greater than an implementation defined limit which are output in NR3 form. This implementation defined limit shall not be less than 999999.

c) Each number that can not be represented exactly as an integer and whose magnitude is greater than or equal to .1 and less than 1000000 is output in signed NR2 form. Each number whose magnitude is less than .1 that can be represented exactly with m significant digits, where m is less than or equal to the number of significant digits being output by the particular implementation, may be output in either signed NR2 form or NR3 form.

d) All other numbers are output in NR3 form.

e) If a number can not be printed completely on the current line, a carriage return and a line feed are output and the number is printed starting in the first character position of the following line.

f) In all cases, trailing zeros to the right of the radix point may be suppressed.

Exceptional Cases

1) Punctuation (a comma or semicolon) is optional in the following situations:

a) before the first print element

b) after the last print element

c) between two print elements, if one and only one of the print elements is a string constant. In this case a comma is implied.

d) between two string constants if there is at least one space character separating them. In this case a comma is implied.

2) The ocurrence of two contiguous quotation marks characters in a string constant will cause one quotation marks character to be output.

3) A print statement with no print elements or punctuation will cause a carriage return and a line feed to be output.

Recognizable Errors

Error messages related to the print elements are described in the section appropriate to the print element.

7.1.3.2 File Input

7.1.3.2.1 READ statements

General Form

READ v-1 , v-2 , v-3 , ... , v-n
where
the v-i are variables (see section 5.)

Description

The execution of a READ statement causes the program to access the data element list which has been constructed from the DATA statements in the program (see section 7.2.1.2) and to assign data element values from that list to the variables specified in the READ statement. The order of appearance of the variables in the READ statements and the order of occurrence of data elements in the data element list determines the order of association of the variables and the data element values. Following the accessing of each data element, the data element pointer (see section 7.2.1.2) is moved to the next data element. Prior to the execution of the first statement in the program (and prior to the execution of the logically first READ statement) the data element pointer indicates the first item in the data element list. The pointer may be reset to this position, to permit repeated accesses to the data element list, by the execution of the RESET (or RESTORE) statement (see section 7.1.3.2.2).

The mode of the variable in the READ statement and the data element with which it is to be associated by the execution of the READ statement must be the same.

Exceptional Cases

None

Recognizable Error Conditions

When, during the execution of a READ statement, the data element pointer is moved so that its position is past the end of the data element list, a fatal error message shall be output indicating that an attempt was made to read more data than was provided by the program.

Where the data element indicated by the data element pointer is not of the same mode as the corresponding variable in the READ statement being executed (see section 7.1.3.2.1), then a fatal error message shall be output.

7.1.3.2.2 RESET statements

General Syntactic Form

RESET
or
RESTORE

Description
(see also sections 7.1.3.2.1, READ statements and 7.2.1.2, DATA statements).

The data element list which is constructed from the ordered set of DATA statements in the program, has associated with it a data element pointer which indicates the next data element which is available for access by the execution of a READ statement. This data element pointer is set to indicate the first element in the data element list by the execution of a RESET (or RESTORE) statement.

Prior to the execution of the first READ statement in the program, the data element pointer is set to indicate the first element in the data element list.

Exceptional Cases

None

Recognizable Error Conditions

None

7.1.3.3 INPUT statements
General Syntactic Form

INPUT v-1 , v-2 , v-3 , ... , v-n
where

each element v-i is a variable (see section 5.)

Description

When an INPUT statement is encountered by the executing program, a question mark (?) is output to the user's terminal. The execution of the program is then suspended until the user signifies that an input list is ready; this is signalled by the use of a line terminator character being input from the terminal. On receipt of this signal, the program assigns the data elements in the input list to the variables listed in the INPUT statement. It is required that the corresponding elements of the input data list and the variable list in the INPUT statement be of the same mode (numeric or string).

The form of a data element list which is acceptable to an INPUT statement is exactly the same form as that specified for the data element list of a DATA statement (see section 7.2.1.2.)

Exceptional Cases

In order to provide a means to escape from the execution of a program, the implementation shall provide at least one character which when input as part of an input data list shall cause the execution of the program to be halted.

Recognizable Error Conditions

Where the number of items in the input data list does not exactly equal the number of variable elements in the INPUT statement being executed, special actions are to be followed. Where the number of data list elements is less than the number of variable elements in the INPUT statement, the user is requested to provide additional data elements. This condition is signified to the user by a special prompting message. This procedure is to be continued until the requisite number of data elements have been supplied by the user.
Where the number of data list elements is greater than the number of variable elements in the INPUT statement, the user will be informed of the excess. The excess of data elements shall be ignored by the program and the program shall continue execution in the normal manner.
Where the mode of the variable in the INPUT statement and the corresponding data element in the input data list are not in agreement, the program shall issue a non-fatal error message and request the re-input of the total data element list.
Where, during the execution of the INPUT statement, the program is unable to recognize a data element either by fault of incorrect syntax or missing punctuation, the program shall issue a non-fatal error message and the user shall be requested

to re-input the total data element list.

7.1.4 Empty Statements

General Syntactic Form

Zero to one full line of space characters.

Description

The empty statement causes no action to be performed. Execution continues with the next physically sequential statement.

Exceptional Cases

None.

Recognizable Error Conditions

None.

7.2 Non-executable Statements

7.2.1 Declarative Statements

7.2.1.1 Array Declarations

General Form

DIM d-1 , d-2 , d-3 , ... , d-n
where
d-i is the declaration of the naming of a numeric array variable, the declaration of the number of dimensions in the array, and the upper bounds of the indices on each of the dimensions. The general form of a declaration is either a(n-1) or a(n-1 , n-2) where a is an numeric array variable name (see section 5.1.2) and the n-i are unsigned non-negative integer value representation of the upper bounds of the dimensions of the array. The number of dimensions in an array is implicitly specified by the number of bounds declared in each declaration d-i.

Description

The DIM statement explicitly defines one or more numeric array variables and provides the necessary dimensioning information to permit the implementation to allocate storage space for each array and to access the individual elements of the arrays. Where a DIM statement specifies that a numeric array variable is one-dimensional, all subsequent references to elements of that array must contain exactly one subscript expression (see section 5.1.2). Similarly when an array is declared to be two-dimensional, two subscript expressions will be required in each array element reference.

Any number of DIM statements may occur in a program; however a specific numeric array variable name can only occur in at most one DIM statement. DIM statements may occur anywhere within the program.

While the DIM statement specifies explicitly the upper bound on each dimension of a numeric array variable, the lower bound is preset to zero (0) for all implementations and program executions. The dimensions of a numeric array variable cannot be altered during the execution of the program.

The numeric value initially associated with each element of a numeric array, prior to the first reference to any element of the array in an executable statement, shall be zero (0).

Exceptional Cases

Where an element of a numeric array variable is referenced in an executable statement without the explicit appearance of that array name in a DIM statement, the implementation shall supply an implicit declaration which has an equal number of dimensions as the reference has subscript expressions, and upper bounds on each dimension of ten (10). The lower bound shall be set to zero (0) as for explicitly declared numeric array variables.

Declarations of numeric array variable names in DIM statements can use the same names as scalar numeric variables; however, the requirement that all references to elements of the array be expressed syntactically in the form a(s-1) or a(s-1 , s-2) where a is a numeric array variable name and the s-i are subscript expressions, provides a means of distinguishing between the two program elements.

Recognizable Error Conditions

When there exists two or more declarations pertaining to arrays which are assigned the same name, a fatal error message shall be output.

Error conditions resulting from references to array elements are specified in section 5.1.2.

7.2.1.2 DATA Statements

General syntactic Form

DATA c-1 , c-2 , c-3 , ... , c-n

where

each c-i is either a signed numeric representation or a character string. A numeric representation shall conform with the specificatio of American National Standard X3.42-1973 in any one of the allowable forms (signed NR1, signed NR2 or NR3).

The implementation may restrict the representations within limits provided that the following minimum requirements are satisfied:

a) In all numeric representations (signed NR1, signed NR2 or NR3) the implementation shall accept at least six (6) significant digits of precision,

b) In the explicit-point-scaled representation (NR3 form) the implementation shall accept representations which include at least two significant digits in the exrad component, and

c) leading or trailing blanks appended to a numeric representation and within the bounds of the enclosing punctuation (including the end of the DATA statement) shall be ignored.

A character string may be optionally enclosed in quotation marks.

NOTE: the form of the data element list in this DATA statement also satisfies the requirements for the syntax of data element lists which are the input to INPUT statements.

Description

The set of DATA statements in a program together compose a single program data element list, the data elements being ordered with respect to both their order of appearance in their respective DATA statements and by the line number associated with those DATA statements. Operating over this data element list is a data element pointer which indicates the next available data element. This next available data element may be accessed by a READ statement (see section 7.1.3.2.1) and the value represented may be associated with the corresponding variable listed in the READ statement. Following the accessing of the indicated data element, the data element pointer is moved to the next data element in the data element list.

Initially, that is, prior to the execution of the first statement in the program, the data element pointer is set to indicate the first data element in the data element list. The data element pointer may be reset to this initial position by the execution of the RESET (or RESTORE) statement (see section 7.1.3.2.2).

Exceptional Cases

Under the provision that character strings may be expressed in DATA statements without containing them in quotation marks, the rightmost end of a string will be delimited by a comma.

Recognizable Error Conditions

When, during the execution of a READ statement, the data element pointer is moved so that its position is past the end of the data element list, a fatal error message shall be output indicating that an attempt was made to read more data than was provided by the program.

Where the data element indicated by the data element pointer is not of the same mode as the corresponding variable in the READ statement being executed (see section 7.1.3.2.1), then a fatal error message shall be output.

7.2.1.3 User Defined Functions

General Syntactic Form

DEF FNa(p-1 , p-2 , p-3 , ... , p-n) = exp

where

a is an alphabetic character which, combined with the preceding characters FN, compose a unique name for the user defined function,

p-i are parameter .names (see section 5.3.3), and

exp is a numeric expression.

Description

In addition to the implementation supplied functions (see section 8.2) whose specifications for evaluation are provided for the programmer, BASIC permits the user to define new functions which are local to the program in which they are defined. This is accomplished through the use of DEF statements. Once defined user defined functions may be referenced in the same manner as implementation supplied functions (see section 8.1).

The value of the named function over the domain of a set of argument values which replace the occurences of the specified parameters in the definition, is the value of the expression exp.

The definition of user defined functions shall be governed by the following rules:

1) a function definition in terms of a DEF statement may occur anywhere in a program, physically prior to the END statement, irrespective of any reference(s) to the named user defined

function.

2) The expression contained in a DEF statement may not reference the function being defined nor may reference to any other function in that expression either directly or indirectly reference the function being defined.
3) A function (identified by its given name) may be defined only once in a program.
4) The parameters p-i have an existance only within the DEF statement in which they are specified. It is permitted that there exists in the program both other DEF statements which utilize the same parameter names and numeric variables of the same name. Parameters are named with the same naming rules as scalar numeric variables (see section 5.1.1).
5) Variable name references contained within the expression of a DEF statement which do not occur in the parameter list of the same DEF statement, are references to the scalar numeric variables of the program, and are associated with the current values of those variables at each invocation of the function definition.
6) On each evaluation of the function, each occurrence of a parameter p-i is replaced by the value of the corresponding argument in the reference to that function (see section 8.1).
7) A DEF statement must be contained within a single line of text no provision being made for multi-line definitions.

Exceptional Cases

The operands of the defining expression are not constrained to include any or all of the set of parameters p-i of the function.

Recognizable Error Conditions

A fatal error message shall be output when:
1) the name of a parameter, p-i, occurs more than once in the defining parameter list,
2) the defining expression references the function being defined, or a function reference contained within the defining expression references the function being defined either directly or indirectly,
3) a function is defined more than once within the program,
4) the parameter list is empty.

7.2.1.4 END statements

General Syntactic Form

END

Description

The END statement shall be the physically last statement in a program. The END statement serves two purposes within a program; as the physically last statement in the program it is a declarative statement marking the physical end of the program; as an executable statement, the END statement operates in the same manner as a STOP statement (see section 7.1.2.C).

Exceptional Cases

None.

Recognizable Error Conditions

The absence of an END statement in a program shall be cause for the issuance of a non-fatal error message, with the equivalent of the END statement being supplied by the implementation and placed after the physically last statement of the program.

7.2.2 REMark statements

General Syntactic Form

REMARK c
or REM c

where c is an optional character string.

Description

The REMARK statement is used to add comments to a BASIC program and does not affect program execution.

If a GO TO, ON, GOSUB, or IF statement references a REMARK statement, the next physically sequential executable statement following the REMARK statement will be executed.

Exceptional Cases.

None.

Recognizable Error Conditions

None.

8. FUNCTIONS

Functions are provided in the BASIC language to enable the programmer to invoke commonly used arithmetic algorithms. A function reference is a notation for expressing the invocation of the algorithm, and to provide the data elements necessary for execution of the algorithm. The result of the execution of an algorithm is a scalar numeric value which replaces the function reference in the expression in which the function reference is embedded.

Two types of function are provided:
a) Implementation supplied functions (see section 8.2) and
b) User defined functions (see section 8.1 for form of function reference and section 7.2.1.3 for form of definition).

8.1 General Form of Function Reference

General Syntactic Form

name(e-1 , e-2 , e-3 , ... , e-n)
where
name is the name of the function being referenced and
e-i is a function argument which is specified as an arithmetic expression (see section 6.1)

Description

A function reference is a notation for the invocation of a predefined algorithm, into which the argument values (that is, the scalar numeric values which are determined by the evaluation of the arithmetic expressions which are listed in the function reference) are substituted for the parameters (see section 7.2.1.3) which are utilized in the function definition. The result of the evaluation of the function, achieved by the execution of the defining algorithm, is a scalar numeric value which replaces the function reference in the expression in which the function reference is embedded. The number of arguments in a function reference must be equal to the number of parameters specified in the function definition.

Exceptional Cases

None

Recognizable Error Conditions

Where a function reference names a function which is not defined

in a DEF statement within the program (see section 7.2.1.3) and which is not supplied by the implementation, a fatal error message shall be output.

Where the number of arguments provided by the function reference is not in agreement with the number of parameters specified in the function definition (either as specified in the corresponding DEF statement or by the implementation definition of the corresponding implementation supplied function), a fatal error message shall be output.

8.2 Implementation Supplied Functions

General Syntactic Form

The syntax of a reference to an implementation supplied function conforms to the specifications for a general function reference (see section 8.1).
The names of implementation supplied functions are agreed between the implementation and the user, provided that they agree with the general syntactic form for function names (see section 5.3).

Description

The implementation supplied functions define algorithms for the evaluation of arithmetic functions. Each element of the minimum set of functions must be supplied with a single argument which can be any valid arithmetic expression (see section 6.1). The value resulting from the evaluation of a function is a scalar numeric value which replaces the function reference in the expression in which the function reference is embedded during the current evaluation of the embedding expression.

Each implementation shall provide the following minimum set of implementation supplied functions:

TRIGONOMETRIC FUNCTIONS

SIN(X)
Computes the sine function of the argument X; the argument must be expressed in radians.

COS(X)
Computes the cosine function of the argument X; the argument must be expressed in radians.

TAN(X)
Computes the tangent function of the argument X; the argument must be expressed in radians.

ATN(X)
Computes the arctangent (in radians) of the argument X;
the range of the result shall be in the range
-(pi/2) < ATN(X) < (pi/2)
where pi is the ratio of the circumference of a circle to its
diameter.

ARITHMETIC FUNCTIONS

LOG(X)
Computes the natural (or Naperian) logarithm (base e) of
the argument X; the argument must be a positive value
greater than zero (0).

EXP(X)
Computes the value of the expression of the base of
natural (or Naperian) logarithms (e = 2.781828...) raised
to the power of the argument X.

SQR(X)
Computes the square root of the argument X; the value of the
argument shall not be negative.

ABS(X)
Computes the absolute value of the value of the argument X;
this can be accomplished by forcing the sign of the argument
to be positive.

INT(X)
Computes the integer part of the value of the argument X;
the result of evaluating this function is the greatest
positive integer which is less than or equal to the absolute
value of the argument X with the sign of the argument X.

SGN(X)
Computes a result which is a representation of the sign of
the argument X; when X≠0 then SGN(X) = X/ABS(X)
otherwise SGN(X) = 0

RND(X)
Returns a random number in the range 0<RND(X)<1 according
to a uniform statistical distribution on this interval.
Each random number generated may be computed from the previous
execution of the algorithm according to some fixed algorithm.
The random number generator algorithm may be initialized by
specifying an argument value X which is non-zero. Subsequent
references to the function with a zero (0) valued argument
may cause the generation of the next number based on the
previous one.

Exceptional Cases

None

Recognizable Error Conditions

Each algorithm which defines a function must be provided with
individual distinctive error messages, fatal or non-fatal,
which identify the misuses of the individual function.

8.3 User Defined Functions

See section 7.2.1.3 for form of function definition, and
section 8.1 for form of function reference.

9. BASIC PROGRAMS

9.1 Program Constituents and Ordering

General Syntactic Form

program-body
end-statement

where program-body is composed of from zero (0) to many program
statements.
and end-statement is an END statement.

Description

A BASIC program is a set of program statements followed by an END
statement which is executed in a sequential manner begining with
the physically first executable statement and continuing sequentially
until

1) program control is transferred by a GOSUB, GO TO,
ON, IF, or RETURN statement,

2) a fatal error condition exists,

3) the user interupts program execution during
an INPUT operation, or

4) a STOP or END statement is executed.

Exceptional Cases

None.

```
- 1-
- 2-  Recognizable Error Conditions
- 3-
- 4-  A fatal error message shall be output if two or more program
- 5-  statements are labelled with the same line number.
- 6-
- 7-
- 8-              finis
```

COMPUTER AND INFORMATION SCIENCE

TECHNICAL NOTES

The following TECHNICAL NOTES are now available at the Computer and Information Science Department, Graduate Research Center except those marked with an asterisk.

*TN/CS/00001 (Out of Date)	Current Research Toward the Standardization and Formal Definition of PL/I by John A.N. Lee (April 1,1968).
*TN/CS/00002	Discrete Markov Chains: An Heuristic Approach by Sue N. Stidham (July 1,1968).
*TN/CS/00003	The Domelki Syntactic Analysis Algorithm by Susan L. Gerhart (August 28,1968).
*TN/CS/00004	A Survey of Hashing Techniques by John A.N. Lee (September 15,1968).
TN/CS/00005	The Recognition and Use of Null Elements in a Syntax Directed Translator by John A.N. Lee (September 19,1968).
*TN/CS/00006 (Revision is TN/CS/00020)	SYNFUL, A Proposed General Purpose Translator System by John A.N. Lee (October 1,1968).
TN/CS/00007	Sorting Almost Ordered Arrays by Caxton C. Foster (November 18,1968).
*TN/CS/00008 (Out of Date)	Vienna Definition Language--Semantics by John A.N. Lee (December 13,1968).
*TN/CS/00009	An Examination of Two Hash Transforms by Caxton C. Foster (May 9,1969).
TN/CS/00010	Multiplexing Without Tears by Caxton C. Foster (May 30,1969).
*TN/CS/00011 (Out of Date)	Vienna Definition Language--A Generalization of Instruction Definitions by John A.N. Lee and Delmore Wu (April,1969).
*TN/CS/00012	An Unclever Time-Sharing System by Caxton C. Foster (October,1969).
*TN/CS/00013	The Formal Definition of the Basic Language by John A.N. Lee (April,1970). (Also published in Computer Journal and as Technical Report 72B-1)
TN/CS/00014	A Debugging Aid by Caxton C. Foster and Hugh C. Schulz (January 16,1970).
*TN/CS/00015	Conditional Interpretation of Operation Codes by Caxton C. Foster and Robert H. Gonter (February,1970).
TN/CS/00016	Some Simple Algorithms for Content Addressable Memories by Caxton C. Foster (July,1970).
TN/CS/00017	A Data Distributor--The Sprinkler System by Caxton C. Foster (July,1970).

*TN/CS/00018 An Annotated Bibliography on Syntax-Directed Translation by John A.N. Lee, Taveta K. Bogert, and Helen Gigley (July,1970).

TN/CS/00019 Chargoggaggoggmanchaugagoggchaubunagungamaug--A Novel Multiply-by-Three Circuit by Caxton C. Foster, Edward Riseman, Fred Stockton, and Conrad Wogrin (September,1970).

TN/CS/00020 SYNFUL--A General Purpose Translator System and Extensive Modification of Technical Note #00006 by John A.N. Lee and Helen Gigley, edited by Taveta K. Bogert (November,1970).

TN/CS/00021 Maintenance Manual for the UMASS Timesharing Version of SYNFUL by Ronald Lautmann (November,1970).

*TN/CS/00022 Microprogramming: A Design Alternative by Michael J. Sullivan (December,1970).

TN/CS/00023 A Simulated Associative Memory by Caxton C. Foster (December,1970).

TN/CS/00024 A Five Tape Algorithm for the Instant Playback Problem by Caxton C. Foster (December,1970).

TN/CS/00025 A Comparison of Simulation Languages by Albert W. Zukatis (January,1971).

TN/CS/00026 A Stack Oriented Computer by Caxton C. Foster (April,1971).

TN/CS/00027 Certain Formal Properties of the Vienna Definition Language by John A.N. Lee.

*TN/CS/00028 When the Chips are Down by Caxton C. Foster (January,1972).

TN/CS/00029 RAUPEDATA-11 -- A Sophisticated Debugging Program for the PDP-11 by Edward G. Fisher (February,1972).

TN/CS/00030 A Tutorial on Cobol Extensions to Handle Data Bases: The Data Base Group Report by Robert W. Taylor (February,1972).

*TN/CS/00031 System Design: Process Models by Richard H. Eckhouse, Jr. (April,1972).

TN/CS/00032 Automated Accounting Systems by Richard H. Eckhosue, Jr. (April,1972).

TN/CS/00033 A Generalization of AVL Trees by Caxton C. Foster (June,1972).

TN/CS/00034 Conditional Syntactic Specification by J. Dorocak and John A.N. Lee (September,1972).

TN/CS/00035 A Formal Definition of Mini Language Number 7,"Dynamic Type Checking" by Edward G. Fisher (October,1972).

TN/CS/00036 Data Administration, Data Independence, and the DBTG Report by Robert W. Taylor (October 1973).

COMPUTER AND INFORMATION SCIENCE

TECHNICAL REPORTS

The following TECHNICAL REPORTS are now available at the Computer and Information Science Department, Graduate Research Center, except those marked with an asterisk.

70C-2 Organizational Principles for Embryological and Neurophysiological Processes by Michael A. Arbib.

70C-3 On the Likely Evolution of Communicating Intelligence on Other Planets by Michael A. Arbib (August 1, 1970, revised December 30, 1970.)

70C-4 Contextual Error Detection by Roger W. Ehrich and Edward M. Riseman.

70C-5 Transformations and Somatotopy in Perceiving Systems by Michael A. Arbib.

70C-6 Organizational Principles for Theoretical Neurophysiology by Michael A. Arbib, (August 15, 1971).

71B-1 The Definition and Validation of the Radix Sorting Technique by John A. N. Lee, (January, 1972).

71B-2 Two Papers on Group Machines by Michael A. Arbib.

71B-4 Automata with Ranked State Sets by Dieter Schütt.

71C-6 Machines in a Category by M. A. Arbib and E. G. Manes.

72A-1 A Study of the Constraints upon the Parallel Dispatching and Execution of Machine Code Instructions by Caxton C. Foster and Edward M. Riseman.

72B-1 The Formal Definition of the Basic Language by John A. N. Lee.

72B-2 Decomposable Machines and Simple Recursion by Michael A. Arbib and E. Manes.

72C-1 A Contextual Postprocessing System for Error Detection and Correction in Character Recognition by Edward M. Riseman and A. Hanson (October 1972).

73B-1 Adjoint Machines, State-Behavior Machines, and Duality by Michael A. Arbib and Ernest G. Manes (January 1973).

73B-2 Natural State Transformations by Suad Alagić (February 1973). (Revised Nov.1973)

73C-3 A Model of Posited Decisionary and Learning Mechanisms in Mammalian CA-3 Hippocampus by William Kilmer, T. McLardy, and M. Olinski (February 1973).

73C-4 Four Faces of Hal: Using Artificial Intelligence Techniques in Computer-Assisted Instruction by Howard A. Peelle and Edward M. Riseman (March, 1973).

73C-5 System Design of an Integrated Pattern Recognition System, or How to Get the Best Mileage out of your Used Pattern Classifier by A.R. Hanson and E.M. Riseman, (June 1973).

73C-6 Neural Models of Spatial Perception and the Control of Movement, by Michael A. Arbib, C. Curt Boylls & Parvati Dev (June 1973).

73B-3 Foundations of System Theory, I by Michael A. Arbib and Ernest G. Manes, (July, 1973.)

73A-1 ULD and a Description of the PDP-8, by John A. N. Lee (September 1973).

73B-4 Time-Varying Systems, by Michael A. Arbib and Ernest G. Manes (November 1973).

73C-7 Model of a Plausible Learning Scheme for CA3 Hippocampus, by William Kilmer and Melanie Olinski, (Nov., 1973).

73B-5 Algebraic Aspects of Algol 68, by Suad Alagić (November 1973).

73C-8 Biology of Decisionary and Learning Mechanisms in Mammalian CA3-Hippocampus, by William Kilmer (November 1973).

73C-9 Eye Movements and Visual Perception: A "Two Visual System" Model by Richard L. Didday and Michael A. Arbib (December 1973).

73A-1.1 Basic Specifications, by John A.N. Lee, Steven R. Beckhardt, and Arthur I. Karshmer (July 1973).