NBSIR 78-1420-1 R

NBS Minimal BASIC Test Programs - Version 1 User's Manual

Volume 1 - Test System Overview

David E. Gilsinn
Charles L. Sheppard

Systems and Software Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

U.S. DEPARTMENT OF COMMERCE

NATIONAL BUREAU OF STANDARDS

NBSIR 78-1420-1

NBS MINIMAL BASIC TEST PROGRAMS - VERSION 1 USER's MANUAL

Volume 1 - Test System Overview

David E. Gilsinn
Charles L. Sheppard

Systems and Software Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, *Secretary*

Dr. Sidney Harman, *Under Secretary*

Jordan J. Baruch, *Assistant Secretary for Science and Technology*

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Acting Director*

TABLE OF CONTENTS

Abstract

This volume is the first of four volumes that comprise the user's guide to the NBS Minimal BASIC test programs. These programs test the conformance of a processor for the BASIC programming language to the specifications given in BSR X3.60 Proposed American National Standard for Minimal BASIC, which is expected to be a Federal Standard. This volume introduces the test system, and covers test program design considerations, user instructions, a discussion of experience in using the system, as well as some observations on testing as a whole. This volume also contains a cross reference index between specifications within the Minimal BASIC standard and the test programs. Volumes 2 through 4 contain brief descriptions, listings and sample outputs of the individual test programs for Minimal BASIC. The entire set of programs is available on magnetic tape from the Institute for Computer Sciences and Technology at the National Bureau of Standards.

Key Words: BASIC, BASIC standard, BASIC validation, compiler validation, computer programming language, computer standards

Acknowledgements

The authors owe thanks to many people for their assistance during the development of these test routines. We thank those members of the American National Standards Institute subcommittee X3J2(BASIC)who assisted with many of the details of the tests and have given us pointers during the development stage.We acknowledge the help of Prof.J.A.N.Lee of the Virginia Polytechnic Institute and the Laboratory Automation Section of the National Bureau of Standards. Both provided the authors with test programs that contributed in a significant way to this test system. We furthermore acknowledge the leadership in compiler test program development shown by Frances E. Holberton and Elizabeth G. Parker of NBS. The program structure and system design developed by them for the NBS FORTRAN Test Programs [4] assisted the authors organize the standard specifications for testing and develop an orderly sequence of tests. We thank Prof. Marvin Zelkowitz and Mr. Mark Klein of the University of Maryland for providing helpful comments and evaluation of the test routines during the development phase. We thank Mr. Dan Lozier of the Applied Mathematics Division of the National Bureau of Standards for providing us with the test values and arguments for the mathematical function tests. We finally thank Dr. Dennis W. Fife, Susan Mitchell, Erica Jen, Beatrice Marron, Belkis Leong-Hong, and I. Trotter Hardy of the Computer Science Section of the National Bureau of Standards for their comments and assistance during the preparation and editing of these volumes.

Note to the Reader

This documentation is a preliminary review copy of the NBS test programs for the Minimal BASIC computer language. They will be used for Government-wide validation of BASIC processors procured by Federal agencies in order to test conformance with the forthcoming American National Standard for Minimal BASIC. Inasmuch as no validation system has previously been available for BASIC, and the specification of Minimal BASIC is still under consideration for adoption as a Federal and national standard, it is appropriate to distribute the test programs for review and comment by concerned parties. All comments and suggestions on the programs and the documentations should be directed to:

Project Manager
NBS BASIC Test Programs
Systems and Software Division
Institute for Computer Sciences
and Technology
National Bureau of Standards
Washington, D.C. 20234

The programs and the documentations will be reissued with any advisable revisions as soon as possible after the American National Standards Institute formally adopts Minimal BASIC as a national standard.

Future validation program releases will occur, first of all, when more comprehensive or precise tests become available for features of the Minimal BASIC standard. Secondly, new releases will be made after the adoption of any enhancement standard for BASIC.

1.0 Introduction

The NBS Minimal BASIC test system is composed of a collection of relatively short programs written using the Minimal BASIC language and designed to exercise an implementation of Minimal BASIC, i.e. a processor (hardware and software) that translates the language statements to executable form. The programs begin with the least sophisticated PRINT capability and proceed through assignment of variables to control statements and expression construction. From this point the tests move on to loops, data structures, internal subroutines, multiway branches and data input. They end with an examination of the accuracy of the supplied mathematical functions and compound algebraic expressions. The programs are self-contained and are ordered by increasing difficulty. Each of the programs tests one or a few specific features of the Minimal BASIC language. In order to properly interpret the results of the test system, the user must be thoroughly familiar with the proposed American National Standards Institute Standard for Minimal BASIC, BSR X3.60 [1].

The tests are not simply syntax conformance tests but are implementation tests. They were written from the point of view of one who wished to determine the limits of capability of his implementation. For this reason the programs test not only, for example, that algebraic operations are recognized, but that they return accurate results. They further test that exceptional circumstances are recognized and reported when required.

2.0 Test Program Design Considerations

2.1 The Minimal BASIC Language

The Proposed ANSI Minimal BASIC Standard X3.60 presents the form for and the interpretation of programs written in the Minimal BASIC programming language. This subset language of BASIC represents a small nucleus of capabilities that will be followed in the future with one or more compatible enhancement extensions. These extensions will be standardized at a later time. The enhancements are presently being designed as upward extensions to Minimal BASIC in such areas as files, flexible input/output formatting, mathematical functions and matrices, control of real-time processing, string manipulation and other general capabilities.

As a language Minimal BASIC includes capabilities necessary to compute arbitrary arithmetic expressions, assign numeric and string values to variables, accept input, display output, structure programs into simple loops and rudimentary "subroutines," and to allow users to define simple value-returning functions. Minimal BASIC also provides a modest set of "built-in" functions to provide users with sine, tangent, log, square root, pseudo-random numbers, and some other functions.

In general a standard Minimal BASIC program is composed of a sequence of lines, ordered by line numbers, with the last line being an END statement. The program is sequentially executed except for control statement action, exception conditions, or user interruptions. The statements and implementation supplied functions allowed in Minimal BASIC are:

1. Supplied Functions: RND, ABS, ATN, COS, EXP, INT, LOG, SGN, SIN, SQR, TAN

2. Assignment: LET

3. Control Statements: END, FOR - NEXT, GOSUB - RETURN, GOTO, IF - THEN, ON - GOTO, STOP

4. Input-Output: INPUT, PRINT

5. Data Initialization: DATA, READ, RESTORE

6. Data Declarations: DIM, OPTION BASE

7. User-Defined Functions: DEF

8. Miscellaneous Statements: RANDOMIZE, REM

2.2 Test System Logic and Organization

The system of validation tests executes the required specifications of Minimal BASIC in a progressive manner. It is clear, however, that any set of programs which is designed to test complex specifications can never test every interaction of every statement, with all permissible forms, in all permissible positions in an executable program. However, the system was designed so that each statement type was executed at least once and that those parts of the language that had been tested were relatively easy to determine. Secondly, as the tests extended the language capabilities, the later tests depended only on the capabilities already tested in earlier programs. For this reason the test programs are designed to be run in a fixed sequence proceeding from tests of very simple features through those of greater complexity. In some cases, this order was determined by the need for lower level features to be tested first so that they, in turn, could be used to test higher level features. The PRINT statement, for example, is fundamental to all tests, for it is the only means of getting output from a program and thus of verifying any test results. Consequently the PRINT tests were developed first.

Almost as fundamental as getting output is the ability to assign values--both numeric and string--to variables. Several programs test this capability. Once one knows that assignment of values and output of strings and numeric constants both work one can examine the control structures GOTO and IF - THEN.

Testing the GOTO was straightforward, as were tests of IF - THEN with a string comparison. Tests of numeric expressions, however, were found to depend on the ability to subtract two values in order to compare a computed value to a true value and then branch on the absolute value of the result. This requires introducing simple arithmetic expressions and the absolute value function. The absolute value function had to be tested in order to be sure that statements of the form

```
5000 IF ABS(A3) <= 1E-6 THEN 1000
```

would work.

The rest of the programs were developed similarly, although as the language capabilities tested became more sophisticated the order of testing was not as crucial because these latter capabilities were not used in later programs as extensively as the earlier capabilities. Each test routine was developed on the basis of an ad hoc evaluation of each statement type. For example, the transfer of control statements all required testing that transfer to previous lines and following lines was possible. One also had to test the system capability of diagnosing a transfer to an illegal line number. In testing loops, one had to consider the various ways that FOR - NEXT limits could be written. In dealing with expression forms one again had to consider each form in a case by case manner. In all cases the tests execute the syntactic variations for each statement type that would likely be used in practice.

A particularly interesting aspect of the validation routines was the testing for accuracy of mathematical expression evaluation. In general, the standard levies no requirements on an implementation for accuracy, but does state that numbers should be printed in a form that exhibits at least six decimal digits, and recommends that six digits of accuracy be maintained. BASIC users might wish to know whether the six digits they were seeing were in fact reasonably accurate. To make such a determination, all of the implementation supplied functions were tested with numbers computed, with the aid of extended precision routines, to greater than six digits accuracy, and then rounded to six digits. This "true" value was then compared to the value generated by the test programs. The comparison used to test the implementation supplied functions in Volume 4 was made in the following manner:

Assume that the true value of an expression or function, say "y", is represented in normalized form as y = mEa, where

m = the mantissa of y
E = the BASIC exponent indicator (base ten)
a = the exponent
0.1 <= ABS(m) < 1.

Similarly, let the computed value of a test program be z = nEa, where

n = mantissa of z

Then z is an approximation to y, given proper rounding, and one can state that the relative error satisfies the inequality

ABS((z-y)/y) <= (0.5E(a-6))/ABS(y)

when y is non-zero (e.g. Fike [2], p.7). If the rounding is incorrect then the inequality is violated, and one knows that the implementation has failed to compute the expression accurately to at least six decimal places. Note that when y = 0 one can consider two cases: If z = 0 then by definition the relative error is zero; if z is non-zero, then one can interchange the values of z and y. In the programs, scaling is used to avoid the extremes.

With respect to the random number generator RND one test concentrated on a statistical evaluation of the uniformity of a set of random numbers generated. There exist a large number of special random number generator tests (see e.g. Knuth, [3], Vol. 2), so a choice had to be made considering the purpose of the entire set of test routines. Exhaustive testing of RND through a variety of tests was not necessary because the entire set of NBS tests are aimed primarily at exposing major flaws or inconsistencies with the specifications of Minimal BASIC. It seemed sufficient, rather, to use a reasonable test to ascertain global uniformity of the random number sequence. Therefore the system has a test that performs 30 experiments on each of 2000 random samples. The chi-square value of each experiment is computed and a statistical goodness-of-fit test is used to test the 30 chi-square values against the cumulative chi-square distribution.

2.3 Test System Objectives

The general objectives of the test system are to evaluate a BASIC processor's capability of (1) handling syntactically correct programs, (2) interpreting the semantics correctly, and (3)returning the required exception condition notifications. The system consists of programs each of which is a sequence of statement lines, the last of which contains an end-statement and each of which is identified by an allowed keyword. The statement lines are ordered by line numbers. The programs must be executed in sequential order, starting with the first line, until (1) some alternate action is dictated by a control statement, (2) an exception occurs for which there is no recovery procedure, or (3) a stop-statement or end-statement is executed. These are the requirements for standard conforming programs and implementations as given in section 4.4 of the Minimal BASIC standard.

The tests do not determine whether a Minimal BASIC processor could detect all nonstandard programs because a standard conforming processor need not reject all nonstandard programs if it implements a superset of the BASIC language. The set of test programs was constructed in order to determine whether an implementation conforms to the ANSI standard in accordance with the rules laid down in section 1.4 of the standard. According to these one should expect that a standard conforming BASIC language processor would accept and process programs conforming to the standard, that it would detect and process exceptional circumstances, that it would interpret the statement and program semantics properly, and that it would accept as input, manipulate and generate as output numbers of at least the precision and range specified in the standard.

In order to test the semantics in many cases the accuracy with which expressions were computed had to be evaluated. This was understandable because otherwise one could not determine whether a processor would perform simple arithmetic operations, evaluate expressions or return intrinsic function values. The fact that a program may be syntactically acceptable does not necessarily mean that it can do what it is asked to do. The tests therefore include program checks on the accuracy with which expressions are evaluated. These checks are made within the standard recommendation of six decimal digits of precision, even though meeting this objective may not be possible for all systems. However, testing this property adds to the BASIC processor quality control capability of this test system. Second, the implementation supplied functions are sampled for accuracy within the six significant digit criteria, even though the standard does not specify any minimum accuracy. Third, a test of the uniformity of the random number generator function RND, which is a standard required supplied function, has been incorporated in these tests.

2.4 Test System Environment

There are four areas related to the operational environment that the standard does not prescribe. First, there are no bounds placed on the size of a program written in Minimal BASIC. Next, the minimum requirements of an automatic data processing system, which is capable of supporting an implementation of a processor for Minimal BASIC, is not specified. Third, the means by which programs written in Minimal BASIC are executed in the control of a supervisory program are not specified. In particular, the standard does not specify the set of commands used to control the environment in which a BASIC program exists. Finally, the standard does not specify the mechanism by which programs written in Minimal BASIC is converted for machine processing.

Therefore, in order to compensate for the lack of these specifications the tests had to be constrained by some a-priori assumptions. First, the tests are written primarily for an interactive mode of execution since BASIC is primarily an interactive language. This does not mean that they could not be run in batch mode, but in general the tests are principally written for an interactive operating environment and the user, during translation and execution of each program, should be accessible through an input-output terminal. Next, the test programs are structured as a sequence of executable programs, each stored as a separate file on a master tape or, if available, a mass storage unit (disk, drum, etc.). The minimal requirements are a tape unit, processor and devices for input and output. With respect to the program size, each file on the master program tape consists of one executable program of less than 300 lines. Each line of code has been restricted to less than or equal to 72 characters. Furthermore, all of the programs restrict core utilization to less than 5000 words of memory for data processing requirements by the programs.

2.5 Test System Assumptions

The tests do not emphasize the detection by the processor of syntax errors. It assumes that whenever a statement or other program element does not conform with the syntactic rules given, either an error condition exists or the statement or other program element has an implementation defined meaning. There are some syntax tests, but these are labeled in the table of contents of volumes 2 through 4.

All of the test programs satisfy the proposed Minimal BASIC programming conventions with respect to spacing. In particular, the programs all assume that spaces can occur anywhere in a program without affecting the execution of the program except that (1) spaces do not appear at the beginning of a line, (2) within line numbers, (3) within keywords (with the exception of GO TO and GO SUB), (4) within numeric constants, (5) within variable or function names, (6) within multicharacter relational symbols. Furthermore, all keywords in a program are preceded by at least one space and, if not at the end of a line, are followed by at least one space. However, there is a test for spaces appearing within strings. The main reason the conventions were adopted was that no exception conditions for spacing were specified in the standard. Some systems accept the various spacing possibilities above as machine dependent capabilities.

All tests of numeric significance are based on testing six digits of significance only and any numeric constants used have been restricted to falling between 1E-38 to 1E+38. Thus, the tests assume that the environment can handle numbers within this range.

Restricting the program line size to 72 characters made it impossible to test the recognition of a 72 character string constant. But there are tests of the system capability to at least concatenate a 72 character string and display it on an output device. This implies that the output device used with these tests must have a margin of at least 72 characters. The user will find that an output medium with some form of hard copy capability would be useful.

3.0 Index of the Test Programs to the Minimal BASIC Standard

This section cross references the sections of the standard [1] with the test program sections. This is done in Table 1, below. Table 1 consists of three columns, the first identifies the section number of the ANSI Minimal BASIC Standard X3.60. The second column of the table gives a short statement of the specification in that section of the standard. The third column identifies the test program sections that exercise the given specification. Table 2, below, gives the titles of all of the test program files and identifies the volume number in which the tests appear. This table is essentially the combined table of contents of volumes 2 through 4.

TABLE 1
Index of ANSI Standard to Test Programs

Sections of Standard	Specifications of Standard	Test Program Sections
3.2	Legal letters are A through Z.	1.2.1
3.2	Legal digits are Ø through 9.	1.2.1
3.2	Plain-string-characters are letters, digits, asterisk, circumflex, close, colon, dollar, equals, greater-than, less-than, minus, number-sign, open, percent, period, plus, question-mark, semicolon, slant, and underline.	1.2.1
3.2	Remark strings allow all string characters, which are BASIC conforming.	9.1
3.2	Quoted strings can contain blanks, since spaces in quoted strings are significant.	1.2.2
4.4	Leading zeroes have no effect in line numbers.	16
4.4	A line can have up to 72 characters.	1
4.4	An END statement is the physically last statement in a program.	142,143
4.5	An exception condition when two lines have the same line number.	17
4.5	An exception condition when statement lines are out of order.	18
5.1	NR1 or implicit point representation of numeric constants.	4
5.1	NR2 or explicit point unscaled representation of numeric constants.	2
5.1	NR3 or explicit point scaled representation of numeric constants.	5
5.1	Implicit point scaled representation of numeric constants.	5
5.1	A string-constant is a character string enclosed in quotation marks.	1
5.4	In a numeric constant, "E" stands for "times ten to the power".	5

5.4	If no sign follows the symbol "E", then a plus sign is understood.	5
5.4	A program can contain numeric constants which have an arbitrary number of digits, though implementation may round the values of such constants to an implementation-defined precision of not less than 6 significant decimal digits.	69
5.4	The implementation-defined range of a numeric constant shall be at least 1E-38 to 1E+38.	8
5.4	A string-constant has as its value the string of all characters between the quotation marks.	1
5.4	Spaces are not ignored in a string constant in a LET statement.	1
5.4	Spaces are not ignored in a string constant in an IF statement.	14
5.5	An error condition when the magnitude of a nonzero numeric-constant is outside the range of the implementation (nonfatal error--supply zero if the magnitude is too small or machine infinity with the appropriate sign if too large).	61,62
5.6	Strings of only 18 characters long are required to be assignable to string variables.	1
6.1	Simple numeric variables consist of a letter followed by an optional digit.	6
6.1	Arrays are named by a single letter, so that subscripted numeric variables consist of a letter followed by one or two numeric expressions enclosed within parentheses.	39
6.1	String variables consist of a letter followed by a dollar sign.	1
6.1	A dollar sign serves to distinguish string from numeric variables.	1
6.4	Numeric values are associated with numeric variable names.	6
6.4	String values are associated with string variable names.	1
6.4	String variable values can be changed by the execution of statements in the pro-	1

	gram.	
6.4	Numeric variable values can be changed by the execution of statements in the program.	6
6.4	The length of the character string associated with a string-variable can vary during the execution of a program from length of zero characters (signifying the null or empty string) to 18 characters.	1
6.4	A subscripted variable refers to the element in the one- or two-dimensional array selected by the value(s) of the subscript(s).	39
6.4	The value of each subscript is rounded to the nearest integer.	153
6.5	An error condition when the same subscripted variable occurs with different numbers of subscripts.	54
6.5	An error condition when a subscript is not in the range of the explicit or implicit dimensioning bounds.	55,56
7.1	String-expressions are composed of either a string-variable or a string-constant.	1
7.2	Numeric-expressions can be constructed from constants.	19,20,21
7.2	Numeric-expressions can be constructed from variables.	22,23
7.2	Numeric-expressions can use the addition operator on constants.	19
7.2	Numeric-expressions can use the subtraction operator on constants.	20
7.2	Numeric-expressions can use the multiplication operator on constants.	20
7.2	Numeric-expressions can use the division operator on constants.	21
7.2	Numeric-expressions can use the involution operator on constants.	21
7.2	Numeric-expressions can use the addition operator on variables.	22
7.2	Numeric-expressions can use the subtraction operator on variables.	22

7.2	Numeric-expressions can use the multiplication operator on variables.	23
7.2	Numeric-expressions can use the division operator on variables.	23
7.2	Numeric-expressions can use the involution operator on variables.	23
7.4	The plus [+] represents addition which adds values.	19
7.4	The minus [-] represents subtraction which subtracts values.	20
7.4	The asterisk [*] represents multiplication which multiplies two values.	20
7.4	The slant [/] represents division which divides two values.	21
7.4	The circumflex [^] represents involution which raises a value to a power.	21
7.4	Unless parentheses dictate otherwise, involutions are performed first, then multiplications and divisions, and finally additions and subtractions.	30,31
7.4	In the absence of parentheses, and where it matters mathematically, operations of the same precedence are associated to the left.	30,31
7.4	Thus, A^B^C is interpreted as (A^B)^C.	30
7.4	In a function reference, the number of arguments supplied must be equal to the number of parameters required by the definition of the function.	116-139
7.4	A function reference is a notation for the invocation of a predefined algorithm, into which the argument value, if any, is substituted for the parameter which is used in the function definition.	19,90 94-105 108-111
7.4	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.	19,90 94-105 108-111
7.4	Spaces can be inserted between the component syntactic elements of numeric-expressions.	33

7.5	There is an error condition if the number of arguments supplied for a function does not agree with the number required.	113 117-139
7.5	There is an error condition if a function referenced in an expression is not implementation-supplied and is not defined in any def-statement.	140
7.5	There is an error condition if the evaluation of an expression results in division by zero (nonfatal--supply machine infinity with the sign of numerator and continue).	59
7.5	There is an error condition if the evaluation of an expression results in an overflow (nonfatal error--supply machine infinity with the algebraically correct sign and continue).	89
7.5	There is an error condition if the evaluation of the operation of involution results in zero being raised to a negative power (nonfatal--supply machine infinity and continue).	52
7.5	There is an error condition if the evaluation of the operation of involution results in a negative number being raised to a non-integral power.	53
7.5	There is an error condition if the evaluation of a string-expression results in a string overflow.	57,93
7.6	If an underflow occurs in the evaluation of a numeric expression, then the value generated by the operation which resulted in the underflow shall be replaced by zero.	61
8.4	The implementation-supplied function ABS with an argument X, ABS(X), returns the absolute value of X.	19
8.4	The implementation-supplied function ATN with an argument X, ATN(X), returns the arctangent of X in radians.	96
8.4	The implementation-supplied function COS with an argument X, COS(X), returns the cosine of X, where X is in radians.	97
8.4	The implementation-supplied function EXP with an argument X, EXP(X), returns the exponential of X, the value of the base of natural logarithms raised to the power	98

	X.	
8.4	If EXP(X) is less than machine infinitesimal, then its value shall be replaced by zero.	100
8.4	The implementation-supplied function INT with an argument X, INT(X), returns the largest integer not greater than X.	90
8.4	The implementation-supplied function LOG with an argument X, LOG(X), returns the natural logarithm of X; X must be greater than zero.	101
8.4	The implementation-supplied function RND does not use an argument, however, it does find the next random number in series.	108-110
8.4	The implementation-supplied function SGN with an argument X, SGN(X), returns the sign of X.	90
8.4	The implementation-supplied function SIN with an argument X, SIN(X), returns the sine of X, where X is in radians.	104
8.4	The implementation-supplied function SQR with an argument X, SQR(X), returns the nonnegative square root of X; X must be nonnegative.	94
8.4	The implementation-supplied function TAN with an argument, TAN(X), returns the tangent of X, where X is in radians.	105
8.5	There is an error condition if the value of the argument of the LOG function is zero or negative.	102-103
8.5	There is an error condition if the value of the argument of the SQR function is negative.	95
8.5	There is an error condition if an incorrect number of arguments has been supplied for a function reference.	117-137
8.5	There is an error condition if the evaluation of the tangent or exponential function results in an overflow (nonfatal--supply machine infinity with the appropriate sign and continue).	99,107
9.2	The LET statement assigns a numeric expression to a simple numeric variable.	6

9.2	The LET statement assigns a string expression to a string variable.	1
9.2	The LET statement assigns a numeric expression to an array variable.	39
10.1	The GOTO statement allows for an unconditional transfer.	9
10.1	The IF-THEN statement allows for a conditional transfer.	11
10.1	The GOSUB statement gives control to a subroutine.	66
10.1	The RETURN statement returns control from a subroutine.	66
10.1	The ON-GOTO statement allows control to be transferred to a selected line.	70
10.1	The STOP statement allows for program termination.	141
10.2	Any number of spaces can be between the GO and the TO of the goto-statement.	9
10.2	A numeric expression can be used by the ON GOTO statement as its variable.	154
10.4	A goto-statement indicates that execution of the program is to be continued starting at the specified line-number.	9
10.4	If the value of the relation-expression in an if-then-statement is true, then execution of the program continues from the specified line-number; if the relation-expression is false, then execution is continued in sequence.	11
10.4	The relation "less than or equal to" is denoted by <=.	11
10.4	The relation "greater than or equal to" is denoted by >=.	11
10.4	The relation "not equal to" is denoted by <>.	11
10.4	The relation of equality holds between two strings if and only if the two strings have the same length and contain identical sequences of characters.	14
10.4	The expression in an on-goto-statement is evaluated and rounded to obtain an inte-	72

	ger.	
10.4	The rounded expression value in an on-goto-statement is used to select a line-number from the list following the GOTO.	70
10.4	The line numbers in the list of the on-goto-statement are indexed from left to right, starting with 1.	70
10.5	There is an error condition if the integer obtained as the value of an expression in an on-goto-statement is less than one or greater than the number of line-numbers in the list.	72,73
10.5	There is an error condition if a goto-statement points to a nonexistent line-number.	71
10.5	There is an error condition if an if-statement points to a nonexistent line-number.	15
10.5	There is an error condition if a gosub-statement points to a nonexistent line-number.	67
10.5	There is an error condition if an on-goto-statement points to a nonexistent line.	71
10.5	There is an error condition if an attempt is made to execute a return-statement with out having executed a corresponding gosub-statement.	68
11.1	The STEP clause in the for-statement is optional.	34
11.4	For-blocks can be nested, i.e. one can contain another.	34
11.4	In the absence of a STEP clause in a for-statement, the increment is assumed to be +1.	34
11.4	The execution of a for-statement steps the initial value to the limit in increments as specified by the STEP clause.	34
11.5	There is an error condition if a for-statement occurs without a matching next-statement.	36
11.5	There is an error condition if a next-statement occurs without a matching for-statement.	37

11.5	There is an error condition if two for-blocks are interleaved.	38
11.6	The value of the control-variable upon exit from a for-block via its next-statement is the first value not used.	35
11.6	If exit is via a control statement, the control-variable retains its current value and the for-block remains active.	35
12.4	Positive numbers when printed have a leading space.	4,5,6
12.4	Negative numbers when printed have a leading minus sign.	4,5,6
12.4	All numbers when printed have a trailing space.	4,5,6
12.4	The possible print formats for the decimal representation of a number are the same as those described for numeric constants.	4,5,6
12.4	On output, there is a significance-width d to control the number of significant decimal digits printed in numeric representations.	4,5,6
12.4	There is also an exrad-width e to control the number of digits printed in the exrad component of a numeric representation.	5,6
12.4	The value of d shall be at least six.	4,5,6
12.4	The value of e shall be at least two.	5,6
12.4	Each number that can be represented exactly as an integer with d or fewer decimal digits is output using the implicit point unscaled representation (NR1).	4
12.4	Numbers which are not integers, but which have absolute values in the range 0.1-0.5*10^(-d-1) to 10^d-0.5, are to be represented in explicit point unscaled notation (NR2) with d significant decimal digits and a period.	4
12.4	Numbers with absolute values less than 0.1-0.5*10^(-d-1) which can be expressed exactly with d decimal digits following a period are to be represented using the explicit point unscaled notation (NR2) also.	4
12.4	All other numbers are to be represented in the explicit point scaled notation (NR3),	5,6

	sign significand E sign integer, where the value x of the significand is in the range 1<=x<10 and is to represented with exactly d digits of precision, and where the exrad component has one to e digits.	
12.4	String-expressions are evaluated to generate the corresponding string of characters.	1
12.4	The evaluation of the semicolon separator generates the null string.	1
12.4	A null print list will skip a line.	1
12.4	Each print-line is divided into a fixed number of print zones.	1
12.4	All print zones, except possibly the last one on a line, shall have the same length.	1
12.4	This length shall be at least d+e+6 characters in order to accomodate the printing of numbers in explicit point scaled notation (NR3).	1,5
12.4	The argument of the tab-call is evaluated and rounded to the nearest integer n.	150
12.4	If n is greater than the margin m, then n is reduced by an integral multiple of m so that it is in the range 1<=n<=m.	92
12.4	If the columnar position of the current line is less than or equal to n, then enough spaces are generated to set the columnar position to n; if the columnar position of the current line is greater than n, then an end-of-print-line is generated followed by enough spaces to set the columnar position of the new current line to n.	1
12.4	The evaluation of the comma separator generates enough spaces to fill out the current print zone, unless this is the last print zone on the line, in which case an end-of-print-line is generated.	1
12.4	If the evaluation of any print-item in a print-list would cause the columnar position of a nonempty line to exceed the margin, then an end-of-print-line is appended to the string of characters being generated before the characters generated by that print-item. If the evaluation of any print-item generates a string whose length is greater than the margin, then an end-of	91

	-print-line is generated each time the columnar position of the current line exceeds the margin.	
12.4	A completely empty print-list will generate an end-of-print-line, thereby completing the current line of output.	1
12.4	The tab-call places the next character for output in the column specified by its argument.	1
12.5	There is an error condition if the rounded argument of a tab-call is less than one (nonfatal--supply a value of one and continue).	2
13.1	Input-statements provide for interaction with a running program by allowing variables to be assigned values that are supplied from a source external to the program.	82
13.4	After validation of an input-reply supplied during the execution of a program, an input-statement causes the variables in the variable-list to be assigned, in order values from the input-reply.	82,84
13.4	In the interactive mode, the user of the program is informed of the need to supply data by the output of an input-prompt.	82
13.4	Execution of the program is suspended until a valid input-reply has been supplied.	82
13.4	If the response to input for a string-variable is an unquoted-string, leading and trailing spaces are ignored.	83
13.4	Subscript expressions in the variable-list are evaluated after values have been assigned to the variables preceding them (i.e., to the left of them) in the variable-list.	83
13.5	There is an error condition if the type of a datum does not match the type of the variable to which it is to be assigned (nonfatal--allow the input-reply to be resupplied).	85
13.5	There is an error condition if there is too much data in the data-list (nonfatal--allow the input-reply to be resupplied).	86
13.5	There is an error condition if there is	87

	insufficient data in the data-list (non-fatal--allow the input-reply to be resupplied).	
13.5	There is an error condition if the conversion of a numeric datum causes an underflow (nonfatal--allow the input-reply to be resupplied).	88
13.5	There is an error condition if the conversion of a numeric datum causes an overflow (nonfatal--allow the input-reply to be resupplied).	89
13.5	There is an error condition if the conversion of a string datum causes a string overflow (nonfatal--allow the input-reply to be resupplied).	93
14.1	The data-statement provides for the creation of a sequence of representations for data elements for use by the read-statement.	74
14.1	The read-statement assigns numeric variables to numeric information in the data-statement.	74
14.1	The read-statement assigns string information from the data-statement to string variables.	74
14.1	The restore-statement allows the data in the program to be reread.	81
14.4	Data from the totality of data-statements in the program are collected into a single data sequence.	74
14.4	If the execution of a program reaches a line containing a data-statement, it proceeds to the next line with no other effect.	74
14.4	The read-statement causes variables in the variable-list to be assigned values, in order, from the data sequence.	74
14.4	Each time a read-statement is executed, variables in the variable-list are assigned values from the data sequence beginning with the datum indicated by the pointer, and the pointer is advanced to point beyond the data used.	74
14.4	Subscript expressions in the variable-list are evaluated after values have been as-	74

	signed to the vaiables preceding them (i.e., to the left of them) in the list.	
14.4	If the conversion of a numeric datum causes an underflow, then its value shall be replaced by zero.	78
14.5	There is an error condition if the variable-list in a read-statement requires more data than are present in the remainder of the data sequence.	75
14.5	There is an error condition if a string datum does not match the type of the numeric variable to which it is to be assigned.	76
14.5	There is an error condition if the conversion of a string datum causes a string overflow.	77
14.5	There is an error condition if the conversion of a numeric datum causes an overflow (nonfatal--supply machine infinity with the appropriate sign and continue).	79
15.1	The dimension-statement is used to reserve space for arrays.	39
15.1	By use of a dimension-statement, the subscript(s) of an array may be declared to have an upper bound other than ten.	39
15.1	By use of an option-statement, the subscripts of all arrays may be declared to have a lower bound of one.	63,64
15.4	Each array-declaration occurring in a dimension-statement declares the array named to be either one or two dimensional according to whether one or two bounds are listed for the array.	39
15.4	Arrays that are not declared in any dimension-statement are declared implicitly to be one or two dimensional according to their first use in the program, and have subscripts with a maximum value of ten.	39
15.4	The option-statement declares the minimum value for all array subscripts.	63
15.4	If no option-statement occurs in a program, this minimum is zero.	39
15.4	If the execution of a program reaches a	39

	line containing a dimension-statement or an option-statement, it proceeds to the next line with no other effect.	
15.5	There is an error if an upper bound of zero is specified in a dimension-statement for a subscript when an option-statement specifies that all lower bounds are one.	63
16.2	In one program you can have upto 26 user-defined functions (i.e., DEF FNA through DEF FNZ).	112
16.4	Evaluation of functions are performed with respect to argument values of the referencing functions.	111
16.4	The evaluation of the argument list of a function is performed first when used in an expression.	111
16.4	The number of arguments must correspond exactly to the number of parameters within the user-defined function.	138,139
16.4	An argument is not needed if there is not any parameter-list with the user-defined function.	113
16.4	The function within an expression which references a user-defined function should substitute the evaluated value of its argument-list for the parameter-list of that defined function.	111
16.4	Once the defined function receives a value for its parameter-list, it uses this value within its defined expression and substitutes the results as the value of the function within the referencing expression.	111
16.4	The parameter appearing in the parameter-list of a function definition is local to that definition.	111
16.4	A function definition can refer to other defined functions.	111
16.5	There is an error condition if the same function is defined more than once.	114
16.5	There is an error condition if a function is referenced inside its definition.	115
17.1	The randomize-statement overrides the implementation-predefined sequence of pseudo	109

	random numbers as values for the RND function, allowing different (and unpredictable) sequences each time a given program is executed.	
17.6	The RND function in the absence of a randomize-statement will generate the same sequence of pseudo-random numbers each time a program is run.	108
18.1	The remark-statement allows program annotation.	9
18.4	The remark-statement has no effect when executed.	9
18.4	The remark-statement can serve as a statement to which control is transferred.	9

Table 2
Test Programs

Program	Title	Volume
1	Output and Assignment of Strings	2
2	Exception Test for Printing TAB Beyond the Left Margin	2
3	Using Empty Print Items to Space Over Print Zones	2
4	Printing Integer and Fixed Point Constants	2
5	Printing Floating Point Constants	2
6	Printing of Floating Point Numbers (Cont.) and Assignment of Integer and Fixed Point Values	2
7	Assignment of Floating Point Constants	2
8	Testing the Minimal Limits in Magnitude of Numerical Constants	2
9	The REM and GOTO Statements	2
10	Test for GOTO with Illegal Statement Label	2
11	The IF-THEN Statement Used to Compare Positive Numerical Constants	2
12	The IF-THEN Statement Used to Compare Negative Numerical Constants	2
13	IF-THEN Comparison of Negative Constant (Cont.) and Variables Avoiding Parentheses	2
14	Comparing Quoted Strings and String Variables	2
15	Test of IF-THEN Transfer to Illegal Line Number	2

16	Line Labels With and Without Leading Zeroes	2
17	Order of Lines - Two Lines with the Same Line Number	2
18	Order of Lines - Lines Out of Order	2
19	The ABS Function and Elementary Numerical Expressions using Constants	2
20	Elementary Numerical Expressions Using Constants (Continued)	2
21	Elementary Numerical Expressions Using Constants (Continued)	2
22	Elementary Expressions Using Simple Variables	2
23	Elementary Expressions Using Simple Variables (Continued)	2
24	Elementary Operations on Mixed Type Constants	2
25	Elementary Operations on Mixed Type Constants (Continued)	2
26	Elementary Operations on Variables Assigned Mixed Type Constants	2
27	Elementary Operations on Variables Assigned Mixed Type Constants (Continued)	2
28	Addition of Three or More Terms	2
29	Multiplication of Three or More Factors	2
30	Hierarchy of Operators and Parentheses	2
31	Hierarchy of Operators and Parentheses (Continued)	2
32	Evalustion of Expressions Having a Variety of	2

	Type Constants (Continued)	
47	Using Elementary Operations on Subscripted Variables Assigned Mixed Type Constants (Continued)	3
48	Addition of More Than Two Terms Containing Array Elements	3
49	Multiplication of More Than Two Terms	3
50	Hierarchy of Operators and Parentheses	3
51	Evaluation of Expressions that have a Variety of Operators	3
52	Exception Test - Zero Raised to a Negative Power	3
53	Exception Test - A Negative Number Raised to a Non - Negative Power	3
54	Semantic Test - Subscripted Variable with Different Numbers of Subscripts	3
55	Exception Test - A Subscript is not in the Range of the Implicit Dimensioning Bounds	3
56	Exception Test - A Subscript is not in the Range of an Explicitly Dimensioned Variable	3
57	Attempting String Overflow by Variable Assignment	3
58	Test for Undefined Variables	3
59	Exception Test - On Division By Zero	3
60	Exception Test - On Expression Evaluation Resulting in Overflow	3

112	Testing All Possible Parametrized User Defined Function Specifications	4
113	User Defined Functions without a Parameter List	4
114	User Defined Function Diagnostic - The Same Function is Defined More Than Once	4
115	A User Defined Function is Referenced Inside Its Own Definition	4
116	Syntax Diagnostic - Argument List Used Incorrectly with a Defined Function	4
117	Syntax Diagnostic - No Argument List for the ABS Function	4
118	Syntax Diagnostic - No Argument List with ATN	4
119	Syntax Diagnostic - No Argument List with COS	4
120	Syntax Diagnostic - No Argument List with EXP	4
121	Syntax Diagnostic - No Argument List with INT	4
122	Syntax Diagnostic - No Argument List with LOG	4
123	Syntax Diagnostic - No Argument List with SGN	4
124	Syntax Diagnostic - No Argument List with SIN	4
125	Syntax Diagnostic - No Argument List with SQR	4
126	Syntax Diagnostic - No Argument List with TAN	4
127	Syntax Diagnostic - A Defined Function with an Argument but Called without It	4

128	Syntax Diagnostic - ABS Used with too Many Arguments	4
129	Syntax Diagnostic - ATN with too Many Arguments	4
130	Syntax Diagnostic - COS with too Many Arguments	4
131	Syntax Diagnostic - EXP with too Many Arguments	4
132	Syntax Diagnostic - INT with too Many Arguments	4
133	Syntax Diagnostic - LOG with too Many Arguments	4
134	Syntax Diagnostic - SGN with too Many Arguments	4
135	Syntax Diagnostic - SIN with too Many Arguments	4
136	Syntax Diagnostic - SQR with too Many Arguments	4
137	Syntax Diagnostic - TAN with too Many Arguments	4
138	Syntax Diagnostic - Defined Function with more than One Argument	4
139	Syntax Diagnostic - A Defined Function with Illegal Argument	4
140	Syntax Diagnostic - Reference to an Undefined Function	4
141	Testing the STOP Statement	4
142	END Statement must be the Last Program Statement	4
143	A Program Requires an END Statement	4
144	Compound Expressions as Arguments for ABS and ATN	4

145	Compound Expressions as Arguments for COS and EXP	4
146	Compound Expressions as Arguments for LOG and SGN	4
147	Compound Expressions as Arguments for SIN and SQR	4
148	Compound Expressions as Arguments for TAN and INT	4
149	Compound Expressions as Arguments for a User Defined Function	4
150	Compound Expressions Used as PRINT Items and TAB Arguments	4
151	Compound Expressions Used in IF-THEN Statements	4
152	Compound Expressions Used in FOR-NEXT Statements	4
153	Compound Expressions Used in Subscripts	4
154	Compound Expression Used in an ON-GOTO	4
155	Compound Expressions Using Supplied Functions	4
156	Semantic Error - Upper Bound of Array set to Zero with Option	4
157	Semantic Error - Multiple Option Statements	4
158	Semantic Error - Option Statement after Array Reference	4
159	Semantic Error - Array Declaration Out of Order	4
160	Semantic Error -	4

	Dimensioning an Array More than Once	
161	Initializing String Variables	4

4.0 User Instructions

The test programs are provided on standard 1/2 inch magnetic tape in a format described separately from this document. It is assumed that the user will accomplish any translation of code set and transfer between recording and storage media that is necessary for the system to be tested; such transformations cannot be specified here. Each program is a separate file on the tape and it will be convenient usually to store each as a separate file on the system to be tested. Any other assumptions have been discussed previously in sections 2.4 and 2.5. Specific procedures for use of the NBS Test Programs for validating BASIC processors for Federal government use are issued separately by NBS.

4.1 Order of Test Execution

The programs are organized so that they may be executed sequentially from test number 1 to test number 161. This is how a user should perform them, since test failures in early tests might explain possible test failures later.

4.2 User Interaction

Except for the programs that test errors and user interaction through the INPUT statement, none of the test routines require user intervention unless some error is detected in the processor during the code execution. The codes that test for error conditions are labelled as either syntax or semantics tests in the table of contents of the other three volumes to this set. The codes that require user interaction with the program have prompt messages that the user should follow carefully. These messages specify the form and items that should be entered by the user at the time of the input prompt. User interaction is required beginning with test program 82 and ending with program 93.

5.0 Experience

The validation system has been run on a PDP-10 and a UNIVAC 1108. Some of the routines have been used on an IBM 5100, an INTERDATA and a NOVA system. None of these systems lays claim to conforming to the Minimal BASIC, since that standard is still pending within ANSI, and could be changed before official adoption.

All of the tested implementations illustrated some variance from the pending standard, as was expected. Most variances were relatively inconsequential, such as the TAB function printing in the column past the one specified by the standard. Others may take more work to correct, such as the one implementation that could not process strings of 18 characters in length. The most interesting discoveries, however, were two: One implementation was using a new release of its BASIC language processor, and the new version--unlike the old one--would not accept a PRINT statement without an expression list. Another implementation flagged as illegal a numeric constant in the source programs that it could both generate by a separate computation and print without difficulty.

6.0 Observations on Testing

To be completely thorough, validation routines would have to test all possible programs. This is impossible; yet one can try to develop tests that exercise the most useful syntactic variants for each statement type--an achievable objective. Language standards provide a broad specification of language capability, but from the user's point of view, languages cannot be disembodied from their implementations. To be useful, therefore, validation routines must make some test of the implementation-defined features, such as error messages, and the accuracy of evaluation of mathematical expressions.

In general all housekeeping and test control of the routines is written using a fundamental set of language features. In particular, the bulk of the code written was confined to PRINT, LET, GOTO and IF-THEN. Although the authors did not always confine themselves to these statements, preferring instead to build on new features as they were tested in turn, in retrospect it might have been wise to do so. The present routines, for example, would be difficult to run on an implementation that failed several early tests, thereby rendering later tests unusable or unreliable. By confining the code to a very elemental subset of capabilities, however, one perhaps could have made it feasible to test higher level features even when many other features failed, so long as the tested implementation supported the elemental subset properly. The difficulty with this approach, however, is that the validation routines become longer and less readable than at present. The design that has been carried out we believe is a reasonable and workable compromise on this issue.

7.0 References

[1] Proposed American National Standard for Minimal BASIC, X3.60, American National Standards Institute, New York, May 1977.

[2] C. T. Fike, Computer Evaluation of Mathematical Functions, Prentice-Hall, Englewood Cliffs, New Jersey (1968).

[3] D. E. Knuth, The Art of Computer Programming, Vol. 2, Addison-Wesley Publishing Company, Reading, Massachusetts (1969).

[4] F. E. Holberton, E. G. Parker, "NBS FORTRAN Test Programs," NBS Special Publication 399, 3 Vols., U. S. Government Printing Office, Washington, D. C. 20402

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS IR 78-1420-1	2. Gov't Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE NBS Minimal BASIC Test Programs - Version 1 User's Manual Volume 1 - Test System Overview			5. Publication Date January 1978
			6. Performing Organization Code
7. AUTHOR(S) David E. Gilsinn and Charles L. Sheppard			8. Performing Organ. Report No.
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234			10. Project/Task/Work Unit No. 6401121
			11. Contract/Grant No.
12. Sponsoring Organization Name and Complete Address *(Street, City, State, ZIP)* National Bureau of Standards			13. Type of Report & Period Covered Final
			14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)*

This volume is the first of four volumes that comprise the user's guide to the NBS Minimal BASIC test programs. These programs test the conformance of a processor for the BASIC programming language to the specifications given in BSR X3.60 Proposed American National Standard for Minimal BASIC, which is expected to be a Federal Standard. This volume introduces the test system, and covers test program design considerations, user instructions, a discussion of experience in using the system, as well as some observations on testing as a whole. This volume also contains a cross reference index between specifications within the Minimal BASIC standard and the test programs. Volumes 2 through 4 contain brief descriptions, listings and sample outputs of the individual test programs for Minimal BASIC. The entire set of programs is available on magnetic tape from the Institute for Computer Sciences and Technology at the National Bureau of Standards.

17. KEY WORDS *(six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons)*

BASIC; BASIC standard; BASIC validation; compiler validation, computer programming language; computer standards.

18. AVAILABILITY	19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
☐ Unlimited ☒ For Official Distribution. Do Not Release to NTIS	UNCLASSIFIED	43
☐ Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13 ☐ Order From National Technical Information Service (NTIS) Springfield, Virginia 22151	20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED	22. Price

USCOMM-DC 29042-P74