DOCUMENT RESUME

ED 083 819 EM 011 542

AUTHOR Isaacs, Gerald L.

TITLE Interdialect Translatability of the Basic Programming Language.

INSTITUTION American Coll. Testing Program, Iowa City, Iowa. Research and Development Div.

SPONS AGENCY Office of Education (DHEW), Washington, D.C.
REPORT NO ACT-TB-11
PUB DATE Oct 72
GRANT OEG-0-72-0711
NOTE 18p.

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computer Programs; Computer Science; *Language Standardization; Programing; *Programing Languages; State of the Art Reviews
IDENTIFIERS BASIC; Beginner's All Purpose Symbolic Instruction Code; *Interdialect Translatability

ABSTRACT

A study was made of several dialects of the Beginner's All-purpose Symbolic Instruction Code (BASIC). The purpose was to determine if it was possible to identify a set of interactive BASIC dialects in which translatability between different members of the set would be high, if reasonable programing restrictions were imposed. It was first established that the four programing capabilities of: 1) computational ability and precision, 2) execution of a large-sized program, 3) accession to creation of external files, and 4) generation of formatted output were necessary if complex projects were to be undertaken. It was found that translation of most statements in BASIC was easily accomplished. Operands, relations, names, strings, arrays, functions, input and branching were among these. Difficulties were mainly encountered in file handling, chaining or subroutine calling, and output formatting. Detailed reports on the translatability of these elements have been compiled, a roster of 21 fundamental rules for translatability provided and three categories of dialects identified. These are: 1) BASIC dialects missing one critical element, 2) dialects lacking only formatted output capability, and 3) preferred dialects. Therefore, BASIC translatability is a fact and can be performed easily if a few rules are followed. (PB)

ACT TECHNICAL BULLETIN NO. 11

INTERDIALECT TRANSLATABILITY OF THE BASIC PROGRAMMING LANGUAGE

by

Gerald L. Isaacs

The Research and Development Division
The American College Testing Program
P. O. Box 168, Iowa City, Iowa 52240

October, 1972

Interdialect Translatability of the BASIC Programming Language

by

Gerald L. Isaacs

Introduction

The BASIC (Beginner's All-purpose Symbolic Instruction Code) programming language is a mathematically-based conversational problem-solving language. It has wide application in business, scientific, and educational environments. It is powerful, efficient, flexible, and has the precision necessary for most tasks. Also, its syntax is simple and easy to learn. The BASIC programming language is simple enough so that an inexperienced programmer can use it and has enough power and flexibility so that the experienced programmer can write his programs efficiently. BASIC was first developed under Professors John G. Kemeny and Thomas E. Kurtz at Dartmouth College in 1963-1964. Since then, BASIC has been transformed into more than twenty different major dialects. Each of these transformations has added to or modified the original language.

Due to the many differences among dialects of BASIC, unless care is taken in the initial programming it is both time consuming and difficult to readily translate a program from one dialect to another. However, if a few rules are followed, it may be possible to translate within a large set of dialects with a minimum of effort. In this paper we investigate this possiblity in some detail.

The research reported herein was performed pursuant to Grant No. OEG-0-72-0711 with the Office of Education, U.S. Department of Health, Education, and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official Office of Education position or policy.

The specific purpose of this study is to identify and investigate in detail those BASIC dialects that would form a set in which translatability would be high if reasonable programmin restrictions are imposed. Only BASIC dialects that are interactive have been studied. More than twenty BASIC dialects are examined in this study.

Some of the dialects are immediately transportable from one computer in a manufacturer's line to another, e.g., the XEROX BASIC runs on the Sigma 5, 6, 7, 8, and 9. Also, several BASIC dialects are upward compatible on computers in the same line, e.g., the BASIC dialect on the Hewlett Packard 2000B will run on the 2000C, 2000E, and 2000F. For most dialects, some translation must be done if a program written in the BASIC of one computer is to be run on a second computer. This study was motivated by the desire to produce readily translatable conversational language interactive programs for computer-assisted data analysis and decision making in an educational environment. The conclusions of the study will, however, apply quite generally since the aforementioned applications are very demanding in terms of text handling capability, computational power, and formatting.

Important Programming Capabilities

There are four programming capabilities that should be present if a project of any magnitude or complexity is to be undertaken. The first of these is computational ability and precision. Of the more than twenty dialects examined, all were found to provide at least six digits of accuracy and to support the basic arithmetic operations plus exponentiation. Some dialects provided accuracy of up to 15 or 16 digits. Obviously, dialects with only six digit accuracy will not be useful in many scientific applications. Also, there was a large discrepancy as to the largest and smallest absolute number allowed. The smallest maximum

absolute number was approximately 10^{38} while the largest minimum number was approximately 10^{-38}. For a translatable system, the questions of accuracy and precision will need to be considered carefully. A system can only be translated to dialects that provide the needed accuracy.

A second necessary capability of any dialect is that it has the ability to execute a program of the desired size. This may be accomplished in several ways. One method involves mass partition size. That is, a user is allowed as large a partition as is necessary for his task and is swapped in and out of core with many other users. This method may substantially add to cost and execution time. Further, when this method is used, a system that uses a monitor to sequentially execute several programs is not very feasible, since all the programs and the monitor must remain in core. In these circumstances, the user would load and execute each program independently. Such a procedure results in a tolerable inconvenience.

A second method that is used by many dialects is program chaining. This method allows the user to fit a very large program into a small partition by dividing the program into small segments and executing them separately in logical succession. There are two kinds of program chaining. The first calls for a complete overlay of the program in core, and the second, a chaining in which the user may specify where the overlay may begin.

The third method for accomplishing the execution of a large program is through the use of external subroutine calls. In this procedure, the user calls a subroutine that is maintained as a separate file. After it is executed, its core is released thus allowing additional portions of the program to be called into core without destroying existing code. There are some BASIC dialects such as IBM-CPS-BASIC, NCR-CENTURY-100-BASIC 1,

and DEC-PDP 8-BASIC (except EDUSYSTEM 25 and 50) which only allow a fixed area of core and do not permit the user any of the above options for increasing the size of the program to be executed. These dialects are inadequate for most complicated systems.

The third capability that a BASIC dialect should have is the means for accessing and creating external files. Three levels of file capability are supported by the various BASIC dialects. One group of dialects offer no file support, e.g., IBM-CPS-BASIC (UNIV of IOWA), DEC-PDP 8-BASIC (except EDUSYSTEM 25 and 50), and NCR-CENTURY 100-BASIC 1. Presently, the NCR-CENTURY 200-BASIC has no file capability although it is promised in the near future. A second group of dialects supports only sequentially accessed files. The latter group includes IBM-ITF-BASIC, IBM-CALL/OS-BASIC, HONEYWELL 200-BASIC, CDC 6000-KRONOS-BASIC, CDC 6000-VERSION 2-BASIC, BURROUGHS-B5500-BASIC, DEC-PDP 8-BASIC (EDUSYSTEM 25 and 50), UNIVAC 1100-BASIC, and MULTICOMP-BASICX. A third group of dialects supports both sequential access and random access files. Members of this group are HP2000F-2000E-2000C-2000B-BASIC, HP3000-BASIC, XDS-BASIC, GE MARK II-BASIC, LEASCO-RESPONSE I-BASIC, DARTMOUTH-BASIC, DEC-PDP 10-PDP 11-BASIC, and DATA GENERAL-BASIC. The urgency of the need for random access files varies with the application. However, since some type of file support is needed for nearly all applications, a minimum of sequential access to files is almost a must.

External files are used to store data that are too complicated and time consuming to recompute every time they are needed. Files also are needed to pass data between chained segments of a system; if the whole partition is overlayed. Also, files can be used to store results of computations so that the user may decrease the size of his program. In view of this, the BASIC dialects mentioned in the second and third groups of the previous paragraph are more adequate than the dialects in the first group.

A fourth important capability for a BASIC dialect is its conduciveness to generating formatted output. This is accomplished by means of the PRINT USING statement. This statement allows the user to determine what his output is going to look like. He may specify the number of digits to be outputted, the mode of output, and the column(s) in which the output is to appear. Also, the user may specify carriage control, e.g., number of spaces between lines. Most of these may also be accomplished using a PRINT statement. This is much less efficient, requires more programming, and cannot be accomplished in the case of specifying the number of digits. The PRINT USING statement has different syntax in almost every dialect. Therefore, it should be noted that if the PRINT USING is used, it must be modified when translating from one dialect to another. In some dialects the format to be followed is specified in the PRINT USING statement itself, while in others the format is in an IMAGE, FIELD, or format statement. Some dialects use Fortran format for output, e.g., MULTICOMP-BASICX. Others use an example output line with special characters denoting numeric output. Not all systems have formatted output, e.g., HP2000B-BASIC, HP2000E-BASIC DATA GENERAL-BASIC, CDC 6000-BASIC, and UNIVAC 1100-UBASIC, although there is an extension of the UNIVAC 1100-BASIC which does have formatted output capability. It is felt that a system should have a capability for formatted output. However, if it does not, the PRINT statement can provide many of the features of the PRINT USING command. Although the results may not be usually as appealing as with the PRINT USING statement, they provide a satisfactory alternative.

The translation of most statements in a BASIC dialect will be trivial or no translation will be necessary. Operands, relations, names, strings, arrays, functions, input, and branching can be translated with little effort or time. The three difficulties that will be

encountered are file handling, chaining or subroutine calling, and output formatting. Since there is no exact standard for these areas, a knowledge of the statement formats in these areas can help to minimize the expenditure of time and energy.

Comparison of Elements

Operations and Relations:

All BASIC dialects use the same symbols for addition +, subtraction -, multiplication *, and division /. However, there is no standard operator for exponentiation. Different dialects use the following symbols: **, ↑, ^ . The most frequent symbol used for exponentiation is ↑ . If exponentiation can be avoided, translatability in operands is achieved. The string operation of concatenation is not implemented on all dialects. For those in which it is implemented, ampersand (&), plus (+), comma (,) or CAT$ are used. A few of the dialects such as HP2000B, HP2000C, HP2000E, HP2000F, HP3000, PDP 11, LEASCO, and the UNIVAC 1100 implement the logical operands of AND, OR, and NOT. The PDP 11 and UNIVAC 1100-BASIC dialects also support logical equivalence (EQV and EQU, respectively) exclusive or (EOR and XOR, respectively) and implication (IMP). The logical relations symbols for less than (<), greater than (>), not equal (<>), less than or equal (<=), and equal (=) are standard across all the BASIC dialects except for the UNIVAC 1100-UBASIC dialect which uses LSS for less than, GRT for greater than, NEQ for not equal, LEQ for less than or equal, and EQU for equal. The logical relation greater than or equal (>=) is standard across all BASIC dialects except for the UNIVAC 1100 and HONEYWELL 200-BASIC dialects which use the symbols GEQ and =>, respectively.

Names:

In the BASIC programming language there can be up to five types of variable names. These are numeric variable names, string variable names, integer variable names, and user defined function names. A numeric variable name should be either a letter or a letter followed by a single digit. While the IBM-BASIC dialects allow the special characters of $, @, and # to be used anywhere a letter may be used, and IBM-CPS-BASIC allows a single letter or a letter followed by another letter or a number, for reasons of translatability these conventions should not be used. String variables are used in all BASIC dialects except the NCR-CENTURY 100-BASIC 1 and PDP 8-BASIC (except EDUSYSTEM 25 and 50).

There are two conventions used for string variable names. The first is a letter followed by a $. The second is a numeric name followed by a $. For translatability the first convention, a letter followed by a $, should be used. Integer variable names are only allowed in the PDP 11-BASIC and HP3000-BASIC and should be avoided. Array variable names should be confined to a single letter that has not been used elsewhere. Some dialects allow any numeric name to be an array name and allow the same name to be both an array variable name and a numeric variable name. In the interest of translatability, array variable names should be confined to a single unique letter. User defined function names are standard in all BASIC dialects except the NCR 200-BASIC and PDP 11-BASIC. There are no user defined functions in the NCR 200 dialect. The PDP 11-BASIC allows the user defined function to be FN followed by any numeric variable name. All other BASIC dialects limit a user defined function to FN followed by a single letter. The general convention of FN letter should be used.

Strings:

All BASIC dialects for the DEC-PDP 8-BASIC (except EDUSYSTEM 25 and 50) and NCR-CENTURY 100-BASIC 1 have string handling capabilities. String constants are enclosed in quotes. In all BASIC dialects except IBM-CPS-BASIC, UNIVAC 1100-UBASIC, and XDS-BASIC, double quotes (") may be used. In the exceptions, single quotes (') are used. Therefore, if translation is to take place between dialects that use the different types of string quotes, a user must be sure to change all the quotes. Strings vary in length in the BASIC dialects. The shortest string length is 6 characters and the longest string length is over 32,000 characters. There are two groups of dialects, those that allow a maximum of 6 to 22 characters and those that allow string length greater than or equal to 72. The dialects that provide a string length less than or equal to 22 characters are DEC-PDP 8-BASIC (EDUSYSTEM 25 and 50), BURROUGHS-B5500-BASIC, IBM-CPS-BASIC, IBM-ITF-BASIC, XDS-BASIC, IBM-CALL/360-OS-BASIC, HONEYWELL 200-BASIC, NCR-CENTURY 200-BASIC, and UCSD-B6700-BASIC. Several of the BASIC dialects provide string processing functions from which substrings, positions and lengths may be obtained. It should be noted that these functions are not translatable and should not be used if the system is to be translated. If string handling is not needed, then all BASIC dialects can be considered. But if a long string (greater than 22) is needed, then translatability is limited.

Arrays:

All BASIC dialects allow use of arrays to store data. An array may have, at most, two dimensions in all BASIC dialects except CDC 6600-BASIC 2.0, CDC 6600-KRONOS-BASIC and the HONEYWELL 200-BASIC, which allow three dimensions. All BASIC dialects have some limit on

the number of elements. In IBM-CPS-BASIC the limit is 500 elements per array. But in most BASIC dialects it is limited only by the amount of core that is available. Arrays that do not appear in a dimension (DIM) statement are dimensioned ten, or ten by ten, or ten by ten by ten depending upon use and system, in all dialects except PDP 8-BASIC, and NCR-CENTURY 200-BASIC. In these exceptions, an array must be dimensioned in order for it to be used. Depending upon the dialect, arrays start at zero or one. But in matrix (MAT) operations the zero elements are ignored anyway. All BASIC dialects have the MAT operations addition, subtraction, scaler multiplication, multiplication, transposition, and inversion except the PDP 8-BASIC, NCR-CENTURY 200-BASIC, NCR-CENTURY 100-BASIC 1, and UCSD-B6700-BASIC which do not support MAT operations. Also, there is an identity matrix (IDN), a matrix of all ones (CON) and a zero matrix (ZER) in all dialects that have the MAT commands. All dialects that support the MAT commands also support a form of matrix input and output. In addition, some support a file input and output for matrices. Whether an array is translatable or not depends upon several factors, including program size and partition size. The PDP 11-BASIC allows arrays to reside on disc in what is called their virtual storage. But this is the only dialect that supports a feature like this.

Complex Variables:

Only the HP3000-BASIC dialect allows the use of complex variables. Therefore, this capability should be avoided.

Functions:

BASIC functions are divided into two types. The first type includes all functions permanently resident in the system. All BASIC dialects support the following system functions:

ABS	Absolute value
ATN	Arctangent
COS	Cosine
EXP	Exponentiation
INT	Largest integer
LOG	Common logarithm
RND	Randomization
SGN	Sign
SIN	Sine
SQR	Square root
TAN	Tangent (except IBM-CPS-BASIC).

The preceding system functions can be used freely. The various dialects also support many other functions that should be avoided.

The second type of function is a user defined function. These functions pass one or several arguments depending on the dialect. Also, some dialects allow multiple line definitions. To be truly translatable, only single line definitions that pass at most one variable should be used. All BASIC dialects allow user defined functions except for NCR-CENTURY 200-BASIC.

Branching:

There are four types of statements used in BASIC for branching purposes. The first type of branching statement is the FOR statement. This loops control between the FOR statement and its corresponding NEXT statement until a counter reaches a limit. The format that is used in all BASIC dialects is:

FOR variable = initial value TO limit STEP increment.

NEXT variable

Initial value, increment and limit may be any expression in all BASIC dialects except HP2000B and HP2000C where the initial value is a variable or a constant. In all BASIC dialects the loop works in the following manner:

1) The variable is set equal to the initial value.
2) Test if variable is searched or passed the limit.
   a) Execute loop if limit has not been reached.
   b) Exit loop if limit has been reached.
3) Add increment to variable.
4) Go back to step 2.

In all BASIC dialects loops may be nested, but maximum nesting permitted varies between dialects. If the user picks eight as the deepest loop that can be nested, then the system will be translatable.

The second type of branching statement is the IF statement. There are many forms of the IF statement in the BASIC dialects; but there is one that holds across all dialects. That is:

IF expression logical operator expression THEN line number.

The third type of branching statement is the GOTO statement. There are two forms of this statement, the simple GOTO and the computed GOTO. The computed GOTO is not implemented in all dialects and should be avoided. The simple GOTO is standard in all dialects as:

GOTO line number.

The word GOTO may also be GO TO in some dialects but it is not clear from the manuals which is accepted.

The fourth type of branching statement is the GOSUB statement. Here there are also two forms, the simple GOSUB and the computed GOSUB.

The computed GOSUB is not universal and should be avoided. The simple GOSUB has the following syntax

GOSUB line number.

This form is standard across all BASIC dialects.

Therefore, if the preceding forms of the branching statements are used, the users' system will be translatable in terms of branching.

Input:

In the BASIC programming dialect there are two methods for accepting input. The first method is the READ-DATA statement pair. These two statements are completely translatable across all BASIC dialects. The form of these two statements is:

READ var 1, var 2, ... var n

DATA constant, constant, ... constant.

The only restriction is that in the DATA GENERAL-BASIC, CDC 6000-KRONOS-BASIC, CDC 6000-VERSION 2-BASIC, and the PDP 8-BASIC (except EDUSYSTEM 25 and 50) do not allow string variables or constants in the READ or DATA statements. The next read position in the data list can be reset to the beginning using the RESTORE command in all BASIC dialects except DARTMOUTH-BASIC which uses the RESET statement.

The second method for accepting input is via the INPUT statement. In BASIC the INPUT statement accepts input from the users' terminal. The INPUT statement has the following syntax:

INPUT var 1, var 2, ... var n.

This syntax is constant over all BASIC dialects for this statement. Thus, these statements are easily translatable.

Files:

The least translatable of all the statements are the file handling statements. Different dialects have different methods for handling files. In some dialects the user allocates a file name with a FILE statement, a FILES statement or an ASSIGN statement depending upon the dialect. Other dialects implicitly do this in the OPEN or first access. Backspacing and rewinding of files are allowed in a few dialects. Some dialects read from files with an INPUT statement while others use a READ statement. Also, PRINT and WRITE statements are used for writing into files in different dialects. Some dialects sense for end of file with an IF END statement, others use a NODATA statement, while others use an ENDFILE statement. File names are determined from dialect to dialect and even from installation to installation within a dialect. Therefore, file handling is not directly translatable and the program writer should attend carefully to file input and file output statements when designing translatable programs.

Miscellaneous:

There are several aspects of BASIC that do not fall into any of the above categories. The first of these is the range on line numbers across the different dialects. The maximum range found was from 0 to 99999999. However, all dialects except IBM-CPS-BASIC and the PDP 8-BASIC accept line numbers from 1 to 9999. IBM-CPS-BASIC has a range from 1 to 999 and PDP 8-BASIC has a range from 1 to 2046. Therefore, one should use line numbers only from 1 to 9999 for translatability. Unless either of the two above exceptions are to be used.

Another feature is comments or remarks; these can be fully translatable if the syntax is:

REM message.

Some dialects zero all variables before they are used, but this should not be taken for granted across all the dialects.

Also, certain dialects such as PDP 11-BASIC, and the HONEYWELL 200-BASIC allow multiple statements on a single line. This feature should not be used.

The keyword LET should not be dropped from assignment statements since many of the dialects require it. Also, only one variable should be assigned at a time. The format appears as:

LET var = expression.

The following three statements:

STOP

END

RETURN

are completely translatable when used in the above syntax. Some dialects allow a comment to follow. This should be avoided for reasons of translatability.

Summary of Rules for Translatability

1) Avoid the use of exponentiation if possible or use ↑ in all dialects where it is permitted.
2) Do not use logical arithmetic (OR, AND, NOT, etc.).
3) Use the following logical relations: <, >, <>, =, <=, and >= whenever permitted.

4) Use a single letter or a letter followed by a number for a numeric variable name.

5) Use a single letter followed by a $ for string variable names.

6) Use a unique letter for an array variable name.

7) Use FN followed by a single letter for a user defined function name.

8) Use double quotes (") whenever possible.

9) Decide on what length strings are going to be allowed and translate your system within the group your string length specifies.

10) Avoid string handling system functions.

11) Use at most two dimension arrays.

12) Start arrays at 1.

13) Take advantage of the MAT command where applicable.

14) Only the system functions listed should be used.

15) Use only single line user defined functions.

16) Nest loops at most eight deep.

17) Limit the following statements to the listed format.

FOR variable = variable TO expression STEP expression

NEXT variable

IF expression-operator-expression THEN line number

GOTO line #

GOSUB line #

READ var 1, ...

DATA constant 1, ...

INPUT var 1, ...

STOP

END

RETURN

RESTORE

REM message

LET var = expression

18) Do not use multiple statements on a single line.

19) Line numbers should run from 1 to 9999.

20) Do not expect the system to zero all variables.

21) Avoid integer and complex variables.

Translatable BASIC Dialects

Most of the problems in translating one dialect to another are a matter of changing a keyword or format. These changes can be made to the whole program at one time using the edit features of the system. There are two features that must be changed or at least checked very closely. These are the file handling and formatted output capabilities. These are not difficult changes to make, but must be considered carefully.

It was found that the dialects studied fell into three categories. The first of these categories contains those dialects that are missing a critical element. These are:

DEC-PDP 8-BASIC (except EDUSYSTEM 25 and 50) no files, no capacity for chaining, etc.

IBM-CPS-BASIC (UNIV of IOWA) no file capability.

NCR-CENTURY 100-BASIC no files, no capacity for chaining, etc.

NCR-CENTURY 200-BASIC no file capability at this time.

The second category contains those dialects that only do not have formatted output capability.

BURROUGHS-B5500-BASIC

CDC 6600-BASIC 2.0

CDC 6600-KRONOS-BASIC

DATA GENERAL-BASIC

DEC-PDP 8-BASIC (EDUSYSTEM 25 and 50)

HP2000B-BASIC

HP2000E-BASIC

UCSD-B6700-BASIC (University of California, San Diego)

UNIVAC 1100-UBASIC (Mankato State College)

Also included in this category are those dialects that issue mass storage in place of chaining or external subroutine capability.

IBM-CALL/OS-360-BASIC

IBM-ITF-BASIC

The third category contains those dialects which are preferred.

DARTMOUTH-BASIC

DEC-PDP 10-BASIC

DEC-PDP 11-BASIC

GE MARK II-BASIC

HONEYWELL 200-BASIC

HP2000C-BASIC

HP2000F-BASIC

HP3000-BASIC

LEASCO BASIC

MULTICOMP-BASICX (UNIV. OF MASS., AMHERST, CDC-3600)

XDX-BASIC

Therefore, following the recommended translatability rules, a user should be able to obtain a system that is translatable with a minimum of effort and time within the third category and translatable with greater difficulty and expense in the second category.

The information provided above is a synopsis of several extensive charts comparing the above dialects. These charts are available from the author. All information was obtained from manufacturers manuals and is subject to change. It can clearly be seen that BASIC translatability is a fact and can be performed easily if a few rules are followed.