The ABC compiler

Martin L. Kersten
Gert-Jan Akkerman
Marcel Worring
Edo Westerhuis
Frans Kunst
Ronnie Lachniet

Department of Mathematics and Computer Science.
Free University
Amsterdam

ABSTRACT

This manual describes the programming language BASIC and its compiler included in the Amsterdam Compiler Kit.

INTRODUCTION.

The BASIC-EM compiler is an extensive implementation of the programming language BASIC. The language structure and semantics are modelled after the BASIC interpreter/compiler of Microsoft (tr), a short comparison is provided in appendix A.

The compiler generates code for a virtual machine, the EM machine [[ACM, etc]]. Using EM as an intermediate machine results in a highly portable compiler and BASIC code.
The drawback of EM is that it does not directly reflect one particular hardware design, which means that many of the low level operations available within BASIC are ill-defined or even inapplicable. To mention a few, the peek and poke instructions are likely to be behave errorneous, while line printer and tapedeck primitives are unknown.

This manual is divided into three chapters.
Chapter 1 discusses the general language syntax and semantics.
Chapter 2 describes the statements available in BASIC-EM.
Chapter 3 describes the predefined functions, ordered alphabetically.

Appendix A discusses the differences with Microsoft BASIC.
Appendix B describes all reserved symbols.

SYNTAX NOTATION

The conventions for syntax presentation are as follows:

CAPS	Items are reserved words, must be input as shown.
<>	Items in lowercase letters enclosed in angular brackets are to be supplied by the user.
[]	Items are optional.
...	Items may be repeated any number of times
{ }	A choice between two or more alternatives. At least one of the entries must be chosen.
|	Vertical bars separate the choices within braces.

All punctuation must be included where shown.

1. GENERAL INFORMATION

The BASIC-EM compiler is designed for a UNIX based environment. It accepts a text file with a BASIC program (suffix .b) and generates an executable file, called a.out.

1.1. LINE FORMAT

A BASIC program consists of a series of lines, starting with a positive line number in the range 0 to 32767. A line may consists of more than one physical line on a terminal, but is limited to 1024 characters. Multiple BASIC statements may be placed on a single line, provided they are separated by a colon (:).

1.2. CONSTANTS

The BASIC compiler character set is comprised of alphabetic characters, numeric characters, and special characters shown below.

= + - * / ^ ( ) % # $ \ _
! [ ] , . ; : & ' ? > < \ (blanc)

BASIC uses two different types of constants during processing: numeric and string constants.
A string constant is a sequence of characters taken from the ASCII character set enclosed by double quotation marks.
Numeric constants are positive or negative numbers, grouped into five different classes.

a) integer constants

Whole numbers in the range of -32768 and 32767. Integer constants do not contain decimal points.

b) fixed point constants

Positive or negative real numbers, i.e. numbers with a decimal point.

c) floating point constants

Real numbers in scientific notation. A floating point constant consists of an optional signed integer or fixed point number followed by the letter E (or D) and an optional signed integer (the exponent). The allowable range of floating point constants is 10^-38 to 10^+38.

d) Hex constants

Hexadecimal numbers, denoted by the prefix &H.

e) Octal constants

Octal numbers, denoted by the prefix &O.

1.3. VARIABLES

Variables are names used to represent values in a BASIC program. A variable is assigned a value by assigment specified in the program. Before a variable is assigned its value is assumed to be zero.
Variable names are composed of letters, digits or the decimal point, starting with a letter. Up to 40 characters are significant. A variable name can be followed by any of the following type declaration characters:

% Defines an integer variable

! Defines a single precision variable (see below)

Defines a double precision variable

$ Defines a string variable.

Beside single valued variables, values may be grouped into tables or arrays. Each element in an array is referenced by the array name and an index, such a variable is called a subscripted variable. An array has as many subscripts as there are dimensions in the array, the maximum of which is 11.
If a variable starts with FN it is assumed to be a call to a user defined function.
A variable name may not be a reserved word nor the name of a predefined function. A list of all reserved identifiers is included as Appendix B.

NOTES:
Two variables with the same name but different type is considered illegal.
The type of a variable without typedeclaration-character is set, at it's first occurence in the program, to the defaulttype which is (in this implementation) double precision.
Multi-dimensional array's must be declared before use (see DIM-statement).
BASIC-EM differs from Microsoft BASIC in supporting floats in one precision only (due to EM), eg doubles and floats have the same precision.

1.4. EXPRESSIONS

When necessary the compiler will convert a numeric value from one type to another. A value is always converted to the precision of the variable it is assigned to. When a floating point value is converted to an integer the fractional portion is rounded. In an expression all values are converted to the same degree of precision, i.e. that of the most precise operand.
Division by zero results in the message "Division by zero". If overflow (or underflow) occurs, the "Overflow (underflow)" message is displayed and execution is terminated (contrary to Microsoft).

Arithmetic

The arithmetic operators in order of precedence,a re:

^	Exponentiation
-	Negation
*,/,\\\\,MOD	Multiplication, Division, Remainder
+,-	Addition, Substraction

The operator \\ denotes integer division, its operands are rounded to integers before the operator is applied. Modulus arithmetic is denoted by the operator MOD, which yields the integer value that is the remainder of an integer division.
The order in which operators are performed can be changed with parentheses.

Relational

The relational operators in order of precedence, are:

=	Equality
<>	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

The relational operators are used to compare two values and returns either "true" (-1) or "false" (0) (See IF statement). The precedence of the relational operators is lower then the arithmetic operators.

Logical

The logical operators performs tests on multiple relations, bit manipulations, or boolean operations. The logical operators returns a bitwise result ("true" or "false"). In an expression, logical operators are performed after the relational and arithmetic operators. The logical operators work by converting their operands to signed two-complement integers in the range -32768 to 32767.

NOT	Bitwise negation
AND	Bitwise and
OR	Bitwise or
XOR	Bitwise exclusive or
EQV	Bitwise equivalence
IMP	Bitwise implies

Functional

A function is used in an expression to call a system or user defined function. A list of predefined functions is presented in chapter 3.

String operations

Strings can be concatenated by using +. Strings can be compared with the relational operators. String comparison is performed in lexicographic order.

1.5. ERROR MESSAGES

The occurence of an error results in termination of the program unless an ON....ERROR statement has been encountered.

2. B-EM STATEMENTS

This chapter describes the statements available within the BASIC-EM compiler. Each description is formatted as follows:

syntax Shows the correct syntax for the statement. See introduction of syntax notation above.

purpose Describes the purpose and details of the instructions.

remarks Describes special cases, deviation from Microsoft BASIC etc.

2.1. CALL

syntax CALL <variable name>[(<argument list>)]

purpose The CALL statement provides the means to execute procedures and functions written in another language included in the Amsterdam Compiler Kit. The argument list consist of (subscripted) variables. The BASIC compiler pushes the address of the arguments on the stack in order of encounter.

remarks Not yet available.

2.2. CLOSE

syntax CLOSE [[#]<file number>[,[#]<file number...>]]

purpose To terminate I/O on a disk file. <file number> is the number associated with the file when it was OPENed (See OPEN-statement). Ommission of parameters results in closing all files.

The END statement and STOP statement always issue a CLOSE of all files.

2.3. DATA

syntax DATA <list of constants>

purpose DATA statements are used to construct a data bank of values that are accessed by the program's READ statement. DATA statements are non-executable, the data items are assembled in a data file by the BASIC compiler. This file can be replaced, provided the layout remains the same (otherwise the RESTORE won't function properly).

The list of data items consists of numeric and string constants as discussed in section 1. Moreover, string constants starting with a letter and not containing blancs, newlines, commas, colon need not be enclosed with the string quotes.

DATA statements can be reread using the RESTORE statement.

2.4. DEF FN

syntax DEF FN<name> [(<parameterlist>)]=<expression>

purpose To define and name a function that is written by the user. <name> must be an identifier and should be preceded by FN, which is considered integral part of the function name. <expression> defines the expression to be evaluated upon function call.

The parameter list is comprised of a comma separated list of variable names, used within the function definition, that are to replaced by values upon function call. The variable names defined in the parameterlist, called formal parameters, do not affect the definition and use of variables defined with the same name in the rest of the BASIC program.

A type declaration character may be suffixed to the function name to designate the data type of the function result.

2.5. DEFINT/SNG/DBL/STR

syntax DEF<type> <range of letters>

purpose Any undefined variable starting with the letter included in the range of letters is declared of type <type> unless a type declaration character is appended. The range of letters is a comma separated list of characters and character ranges (<letter>-<letter>).

2.6. DIM

syntax DIM <list of subscripted variable>

purpose The DIM statement allocates storage for subscripted variables. If an undefined subscripted variable is used the maximum value of the array subscript is assumed to be 10. A subscript out of range is signalled by the program (when ACK works) The minimum subscript value is 0, unless the OPTION BASE statement has been encountered.

All variables in a subscripted variable are initially zero.

BUGS. Multi-dimensional arrays MUST be defined. Subscript out of range is left unnotified.

2.7. END

syntax END

purpose END terminates a BASIC program and returns to the UNIX shell. An END statement at the end of the BASIC program is optional.

2.8. ERR and ERL

syntax <identifier name>= ERR
<identifier name>= ERL

purpose Whenever an error occurs the variable ERR contains the error number and ERL the BASIC line where the error occurred. The variables are usually used in error handling routines provided by the user.

2.9. ERROR

syntax ERROR <integer expression>

purpose To simulate the occurrence of a BASIC error. To define a private error code a value must be used that is not already in use by the BASIC runtime system. The list of error messages currently in use can be found in appendix B.

2.10. FIELD

purpose To be implemented.

2.11. FOR...NEXT

syntax FOR <variable>= <low>TO<high>[STEP<size>]
.....
NEXT [<variable>][,<variable>...]

purpose The FOR statements allows a series of statements to be performed repeatedly. <variable> is used as a counter. During the first execution pass it is assigned the value <low>, an arithmetic expression. After each pass the counter is incremented (decremented) with the step size <size>, an expression. Ommission of the step size is intepreted as an increment of 1.
Execution of the program lines specified between the FOR and the NEXT statement is terminated as soon as <low> is greater (less) than <high>

The NEXT statement is labeled with the name(s) of the counter to be incremented.

The variables mentioned in the NEXT statement may be ommitted, in which case the variable of increment the counter of the most recent FOR statement. If a NEXT statement is encountered before its corresponding FOR statement, the error message "NEXT without FOR" is generated.

2.12. GET

syntax GET [#]<file number>[, <record number>]

purpose To be implemented.

2.13. GOSUB...RETURN

syntax GOSUB <line number>
...
RETURN

purpose The GOSUB statement branches to the first statement of a subroutine. The RETURN statement cause a branch back to the statement following the most recent GOSUB statement. A subroutine may contain more than one RETURN statement.

Subroutines may be called recursively. Nesting of subroutine calls is limited, upon exceeding the maximum depth the error message "XXXXX" is displayed.

2.14. GOTO

syntax GOTO <line number>

purpose To branch unconditionally to a specified line in the program. If <line number> does not exists, the compilation error message "Line not defined" is displayed.

remarks Microsoft BASIC continues at the first line equal or greater then the line specified.

2.15. IF...THEN

syntax
IF <expression> THEN {<statements>|<line number>} [ELSE {<statements>|<line number>}]

syntax IF <expression> GOTO <line number> [ELSE {<statements>|<line number>}]

purpose The IF statement is used to make a decision regarding the program flow based on the result of the expressions. If the expression is not zero, the THEN or GOTO clause is executed. If the result of <expression> is zero, the THEN or GOTO clause is ignored and the ELSE clause, if present is executed.
IF..THEN..ELSE statements may be nested. Nesting is limited by the length of the line. The ELSE clause matches with the closests unmatched THEN.

When using IF to test equality for a value that is the result of a floating point expression, remember that the internal representation of the value may not be exact. Therefore, the test should be against a range to handle the relative error.

remarks Microsoft BASIC allows a comma before THEN.

2.16. INPUT

syntax INPUT [;][<"prompt string">;]<list of variables>

purpose An INPUT statement can be used to obtain values from the user at the terminal. When an INPUT statement is encountered a question mark is printed to indicate the program is awaiting data. IF <"prompt string"> is included, the string is printed before the the question mark. The

question mark is suppressed when the prompt string is followed by a comma, rather then a semicolon.

For each variable in the variable a list a value should be supplied. Data items presented should be separated by a comma.

The type of the variable in the variable list must aggree with the type of the data item entered. Responding with too few or too many data items causes the message "?Redo". No assignment of input values is made until an acceptable response is given.

remarks The option to disgard the carriage return with the semicolon after the input symbol is not yet implemented.

2.17. INPUT [#]

syntax INPUT #<file number>,<list of variables>

purpose The purpose of the INPUT# statement is to read data items from a sequential file and assign them to program variables. <file number> is the number used to open the file for input. The variables mentioned are (subscripted) variables. The type of the data items read should aggree with the type of the variables. A type mismatch results in the error message "XXXXX".

The data items on the sequential file are separated by commas and newlines. In scanning the file, leading spaces, new lines, tabs, and carriage returns are ignored. The first character encountered is assumed to be the state of a new item. String items need not be enclosed with double quotes, provided it does not contain spaces, tabs, newlines and commas,

remarks Microsoft BASIC won't assign values until the end of input statement. This means that the user has to supply all the information.

2.18. LET

syntax [LET]<variable>=<expression>

purpose To assign the value of an expression to a (subscribted) variable. The type convertions as dictated in chapter 1 apply.

2.19. LINE INPUT

syntax LINE INPUT [;][<"prompt string">;]<string variable>

purpose An entire line of input is assigned to the string variable. See INPUT for the meaning of the <"prompt string"> option.

2.20. LINE INPUT [#]

syntax LINE INPUT #<file number>,<string variable>

purpose Read an entire line of text from a sequential file <file number> and assign it to a string variable.

2.21. LSET and RSET

purpose To be implemented

2.22. MID$

syntax MID$(<string expr1>,n[,m])=<string expr2>

purpose To replace a portion of a string with another string value. The characters of <string expr2> replaces characters in <string expr1> starting at position n. If m is present, at most m characters are copied, otherwise all characters are copied. However, the string obtained never

exceeds the length of string expr1.

2.23. ON ERROR GOTO

syntax ON ERROR GOTO <line number>

purpose To enable error handling within the BASIC program. An error may result from arithmetic errors, disk problems, interrupts, or as a result of the ERROR statement. After printing an error message the program is continued at the statements associated with <line number>.

Error handling is disabled using ON ERROR GOTO 0. Subsequent errors result in an error message and program termination.

2.24. ON...GOSUB and ON ...GOTO

syntax ON <expression> GOSUB <list of line numbers>
ON <expression> GOTO <list of line numbers>

purpose To branch to one of several specified line numbers or subroutines, based on the result of the <expression>. The list of line numbers are considered the first, second, etc alternative. Branching to the first occurs when the expression evaluates to one, to the second alternative on two, etc. If the value of the expression is zero or greater than the number of alternatives, processing continues at the first statement following the ON..GOTO (ON GOSUB) statement.

When the expression results in a negative number the an "Illegal function call" error occurs.

BUG If the value of the expression is zero or greater than the number of alternatives, processing does NOT continue at the first statement following the ON..GOTO (ON GOSUB) statement.

2.25. OPEN

syntax OPEN {"i" | "o" | "r" } , [#]<file number> , <file-name>

purpose To open <file-name> (filename should be quoted) for input/reading or output. If file is not opened for output it has to be existent, otherwise an "file not found" error will occur.

2.26. OPTION BASE

syntax OPTION BASE n

purpose To declare the lower bound of subsequent array subscripts as either 0 or 1. The default lower bound is zero.

2.27. POKE

syntax POKE <expr1>,<expr2>

purpose To poke around in memory. The use of this statement is not recommended, because it requires full understanding of both the implementation of the Amsterdam Compiler Kit and the hardware characteristics.

2.28. PRINT

syntax PRINT <list of variables and/or constants>

purpose To print constants or the contents of variables on the terminal-device. If the variables or constants are seperated by comma's the values will be printed seperated by tabs. If the variables or constants are seperated by semi-colon's the values will be printed without spaces in between. The new-line generated at the end of the print-statement can be suppressed by a semi-colon at the end of list of variables or constants.

2.29. PRINT USING

purpose To be implemented

2.30. PUT

purpose To be implemented

2.31. RANDOMIZE

syntax RANDOMIZE [<expression>]

purpose To reset the random seed. When the expression is ommitted, the system will ask for a value between -32768 and 32767. The random number generator returns the same sequence of values provided the same seed is used.

2.32. READ

syntax READ <list of variables>

purpose To read values from the DATA statements and assign them to variables. The type of the variables should match to the type of the items being read, otherwise a "Syntax error" occurs. If all data is read the message "Out of data" will be displayed.

2.33. REM

syntax REM <remark>

purpose To include explantory information in a program. The REM statements are not executed. A single quote has the same effect as : REM, which allows for the inclusion of comment at the end of the line.

remarks Microsoft BASIC does not allow REM statements as part of DATA lines.

2.34. RESTORE

syntax RESTORE [<line number>]

purpose To allow DATA statements to be re-read from a specific line. After a RESTORE statement is executed, the next READ accesses the first item of the DATA statements. If <line number> is specified, the next READ accesses the first item in the specified line.

Note that data statements result in a sequential datafile generated by the compiler, being read by the read statements. This data file may be replaced using the operating system functions with a modified version, provided the same layout of items (same number of lines and items per line) is used.

2.35. STOP

syntax STOP

purpose To terminate the execution of a program and return to the operating system command interpreter. A STOP statement results in the message "Break in line ???"

2.36. SWAP

syntax SWAP <variable>,<variable>

purpose To exchange the values of two variables.

BUG. Strings cannot be swapped !

2.37. TRON/TROFF

syntax TRON

syntax TROFF

purpose As an aid in debugging the TRON statement results in a program listing each line being interpreted. TROFF disables generation of this code.

2.38. WHILE...WEND

syntax WHILE <expression>
..... WEND

purpose To execute a series of BASIC statements as long as a conditional expression is true. WHILE...WEND loops may be nested.

2.39. WRITE

syntax WRITE [<list of expressions>]

purpose To write data at the terminal in DATA statement layout conventions. The expressions should be separated by commas.

2.40. WRITE

syntax WRITE #<file number> ,<list of expressions>

purpose To write a sequential data file, being opened with the "O" mode. The values are being writting using the DATA statements layout conventions.

3. FUNCTIONS

ABS(X)	Returns the absolute value of expression X
ASC(X$)	Returns the numeric value of the first character of the string. If X$ is not initialized an "Illegal function call" error is returned.
ATN(X)	Returns the arctangent of X in radians. Result is in the range of -pi/2 to pi/2.
CDBL(X)	Converts X to a double precision number.
CHR$(X)	Converts the integer value X to its ASCII character. X must be in the range of 0 to 257. It is used for cursor addressing and generating bel signals.
CINT(X)	Converts X to an integer by rounding the fractional portion. If X is not in the range -32768 to 32767 an "Overflow" error occurs.
COS(X)	Returns the cosine of X in radians.
CSNG(X)	Converts X to a single precision number.
CVI(<2-bytes>)	Convert two byte string value to integer number.
CVS(<4-bytes>)	Convert four byte string value to single precision number.
CVD(<8-bytes>)	Convert eight byte string value to double precision number.
EOF[(<file-number>)]	Returns -1 (true) if the end of a sequential file has been reached.
EXP(X)	Returns e(base of natural logarithm) to the power of X. X should be less then 10000.0.
FIX(X)	Returns the truncated integer part of X. FIX(X) is equivalent to SGN(X)*INT(ABS(X)). The major difference between FIX and INT is that FIX does not return the next lower number for negative X.
HEX$(X)	Returns the string which represents the hexadecimal value of the decimal argument. X is rounded to an integer using CINT before HEX$ is evaluated.
INT(X)	Returns the largest integer <= X.
INP$(X[,[#]Y])	Returns the string of X characters read from the terminal or the designated file.
LEN(X$)	Returns the number of characters in the string X$. Non printable and blancs are counted too.
LOC(<file number>)	For sequential files LOC returns position of the read/write head, counted in number of bytes. For random files the function returns the record number just read or written from a GET or PUT statement. If nothing was read or written 0 is returned.
LOG(X)	Returns the natural logarithm of X. X must be greater than zero.
MID$(X,I,[J])	Returns first J characters from string X starting at position I in X. If J is omitted all characters starting of from position I in X are returned.
MKI$(X)	Converts an integer expression to a two-byte string.
MKS$(X)	Converts a single precision expression to a four-byte string.
MKD$(X)	Converts a double precision expression to a eight-byte string.
OCT$(X)	Returns the string which represents the octal value of the decimal argument. X is rounded to an integer using CINT before OCTS is evaluated.
PEEK(I)	Returns the byte read from the indicated memory. (Of limited use in the context of ACK)
POS(I)	Returns the current cursor position. To be implemented.
RIGHT$(X$,I)	Returns the right most I characters of string X$. If I=0 then the empty string is returned.

RND(X)	Returns a random number between 0 and 1. X is a dummy argument.
SGN(X)	If X>0 , SGN(X) returns 1. if X=0, SGN(X) returns 0. if X<0, SGN(X) returns -1.
SIN(X)	Returns the sine of X in radians.
SPACE$(X)	Returns a string of spaces length X. The expression X is rounded to an integer using CINT.
STR$(X)	Returns the string representation value of X.
STRING$(I,J)	Returns thes string of length Iwhose characters all have ASCII code J. (or first character when J is a string)
TAB(I)	Spaces to position I on the terminal. If the current print position is already beyond space I,TAB goes to that position on the next line. Space 1 is leftmost position, and the rightmost position is width minus 1. To be used within PRINT statements only.
TAN(X)	Returns the tangent of X in radians. If TAN overflows the "Overflow" message is displayed.
VAL(X$)	Returns the numerical value of string X$. The VAL function strips leading blanks and tabs from the argument string.

APPENDIX A DIFFERENCES WITH MICROSOFT BASIC

The following list of Microsoft commands and statements are not recognized by the compiler.

SPC
USR
VARPTR
AUTO
CHAIN
CLEAR
CLOAD
COMMON
CONT
CSAVE
DELETE
EDIT
ERASE
FRE
KILL
LIST
LLIST
LOAD
LPRINT
MERGE
NAME
NEW
NULL
RENUM
RESUME
RUN
SAVE
WAIT
WIDTH LPRINT

Some statements are in the current implementation not available, but will be soon. These include:

CALL
DEFUSR
FIELD
GET
INKEY
INPUT$
INSTR$
LEFT$
LSET
RSET
PUT

APPENDIX B RESERVED WORDS IN BASIC-EM

The following list of words/symbols/names/identifiers are reserved, which means that they can not be used for variable-names.

ABS	AND	ASC	AS	
ATN	AUTO	BASE	CALL	
CDBL	CHAIN	CHR	CINT	
CLEAR	CLOAD	CLOSE	COMMON	
CONT	COS	CSNG	CSAVE	
CVI	CVS	CVD	DATA	
DEFINT	DEFSNG	DEFDBL	DEFSTR	
DEF	DELETE	DIM	EDIT	
ELSE	END	EOF	ERASE	
ERROR	ERR	ERL	ELSE	
EQV	EXP	FIELD	FIX	
FOR	FRE	GET	GOSUB	
GOTO	HEX	IF	IMP	
INKEY	INPUT	INP	INSTR	
INT	KILL	LEFT	LEN	
LET	LINE	LIST	LLIST	
LOAD	LOC	LOG	LPOS	
LPRINT	LSET	MERGE	MID	
MKI	MKS	MKD	MOD	
NAME	NEW	NEXT	NOT	
NULL	ON	OCT	OPEN	
OPTION	OR	OUT	PEEK	
POKE	PRINT	POS	PUT	
RANDOMIZE	READ	REM	RENUM	
REN	RESTORE	RESUME	RETURN	
RIGHT	RND	RUN	SAVE	
STEP	SGN	SIN	SPACE	
SPC	SQR	STOP	STRING	
STR	SWAP	TAB	TAN	
THEN	TO	TRON	TROFF	
USING	USR	VAL	VARPTR	
WAIT	WHILE	WEND	WIDTH	
WRITE	XOR			