MINC-11

Book 3:
MINC Programming Reference

November 1978

This book provides a reference for all of the MINC BASIC statements and commands, giving a complete description of each. *Book 2: MINC Programming Fundamentals* can be used in conjunction with this book for a more fundamental explanation of each command or statement.

Order Number AA-D800A-TC

MINC-11

VERSION 1.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

form. The table shows, for example, the valid values for arguments and the default conditions assumed when you omit optional arguments. Below is a sample table.

Forms ABS(expression)

Component	*Component Type*	*Component Value*	*Default Condition*
ABS	function name/real	0 to 1.7×10^{38}	required component
expression	numeric expression	any value	required component

This table is the table presented with the ABS (absolute value) function. The components of the forms are presented left to right in the form and top to bottom in the table. ABS is the first component.

The component type is usually "string" or "numeric". Sometimes, as in the case of ABS, the component type is more. ABS is a function name; thus, the component type is "function name". However, the ABS function name also takes on a value that is equal to the absolute value of the argument. Thus, the component type is also a "real value". Therefore, the component type is "function name/real", which shows that ABS is a function name that takes on a real value.

The component value shows the possible values of the component. In the case of ABS, the values are real numbers in the range 0 to 1.7×10^{38}. In other cases a component value might be SY0: or SY1:.

The default condition tells whether the component is required and, if not, what happens if the component is omitted.

Instructions The instructions explain the arguments and default conditions in more detail.

Restrictions The restrictions include discussions of when not to use the command, particular problem situations, and possible solutions to some common problems.

Errors The error messages that can appear in using the command or statement are listed. Whenever possible, the section explains the condition causing the message.

CONTENTS

FIGURES

INTRODUCTION

Book 3: MINC Programming Reference describes the MINC system commands, program statements and functions, and the MINC keypad editor.

The prerequisite reading for this book is *Book 2: MINC Programming Fundamentals*. Book 2 is a tutorial presentation of the material covered in this book. If you are a novice programmer, use Book 2 to learn the fundamentals of MINC programming and terminology. Use this book when you want information organized for reference use or further technical information not present in Book 2.

HOW TO USE THE MANUAL

All material in this book appears in alphabetical order. The topics presented range from commands and statement keywords to conceptual topics. All sections use the same organizational structure for the information.

Purpose — The statement of purpose explains how a command or statement operates or why a conceptual topic is important.

Forms — The form shows the complete syntactic form for commands and statements. The components of the form that are required for MINC to complete the operation are printed in black. The components that are optional and can be omitted are printed in blue. Following the form statement is a table summarizing the meanings of the components of the

form. The table shows, for example, the valid values for arguments and the default conditions assumed when you omit optional arguments. Below is a sample table.

Forms ABS(expression)

Component	*Component Type*	*Component Value*	*Default Condition*
ABS	function name/real	0 to 1.7×10^{38}	required component
expression	numeric expression	any value	required component

This table is the table presented with the ABS (absolute value) function. The components of the forms are presented left to right in the form and top to bottom in the table. ABS is the first component.

The component type is usually "string" or "numeric". Sometimes, as in the case of ABS, the component type is more. ABS is a function name; thus, the component type is "function name". However, the ABS function name also takes on a value that is equal to the absolute value of the argument. Thus, the component type is also a "real value". Therefore, the component type is "function name/real", which shows that ABS is a function name that takes on a real value.

The component value shows the possible values of the component. In the case of ABS, the values are real numbers in the range 0 to 1.7×10^{38}. In other cases a component value might be SY0: or SY1:.

The default condition tells whether the component is required and, if not, what happens if the component is omitted.

Instructions The instructions explain the arguments and default conditions in more detail.

Restrictions The restrictions include discussions of when not to use the command, particular problem situations, and possible solutions to some common problems.

Errors The error messages that can appear in using the command or statement are listed. Whenever possible, the section explains the condition causing the message.

Note that the following error message rarely appears under Errors.

?MINC-F-Syntax error; cannot translate the statement

This message has not been included because it always occurs if you mistype a command. The errors listed under each section do not include typographical errors.

Related — Subjects related to the topic of the section are listed. Whenever possible, this section discusses the nature of the relation between the topics.

References — This section lists references to other books in the MINC document set and, where relevant, to other published sources.

Examples — The examples for each topic range from single-line examples showing the behavior of statements and functions to short program fragments. In some cases, the example refers to a major program example for the topic in Book 2.

REFERENCES

ABORT

Purpose

The ABORT system function results in halting program execution. ABORT has two different effects, depending on the value of its argument. In one case, ABORT has an effect similar to a STOP statement. In the other case, ABORT has an effect similar to a STOP statement followed by a SCR command.

Forms

variable=ABORT(code)

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric variable	unknown	required component
=	none	none	required component
ABORT	function name/none	none	required component
code	numeric expression	0,1	required component

Instructions

Use ABORT to stop program execution.

If the value of the code argument is 0, ABORT has the same effect as STOP but does not print a message as STOP does. If the value of the code argument is 1, ABORT has the same effect as STOP (without the stop message) followed by a SCR command.

The value of the variable is unknown after the function has been executed. You do not need to use the variable, it is only part of the syntactic form of the function.

Restrictions

None.

Errors

No error messages apply to this system function.

Related

SCR The SCR command erases the workspace and names it NONAME.

STOP The STOP statement stops program execution, prints a message showing the statement number of the STOP statement, and preserves the workspace.

References

None.

Examples

None.

ABS

Purpose

The ABS function takes on the absolute value of its argument.

Forms

ABS(expression)

Component	*Component Type*	*Component Value*	*Default Condition*
ABS	function name/real	0 to 1.7×10^{38}	required component
expression	numeric expression	any value	required component

Instructions

Use ABS to obtain the unsigned magnitude of any expression.

Restrictions

None

Errors

No error messages apply to this function.

Related

SGN takes on a value corresponding to the sign of its argument. The following identity expresses the relationship between SGN and ABS:

expression = SGN(expression)*ABS(expression)

References

Book 2, Chapter 8.

Examples

Example PRINT ABS(-3)

Result 3

Example A=ABS(29)

Result A takes the value 29.

APPEND

Purpose

The APPEND command combines a program stored on a diskette file with the program already in the workspace. By using APPEND, you can combine separate programs into a single program in the workspace.

When the stored program is merged with the workspace program, the statement numbers determine the position of the merged statements. For duplicate statement numbers, APPEND erases the workspace version of the statement and inserts the statement from the stored file. All other statements are inserted directly according to the statement number.

Forms

APPEND filespec

Component	*Component Type*	*Component Value*	*Default Condition*
APPEND	command	none	required component
filespec	characters	dev:name.typ	prompts for filespec

Instructions

Use the APPEND command whenever MINC displays READY. When you complete one of the two forms, MINC merges the file you specify and the program currently in your workspace, statement number by statement number. When the new file has a statement number that is the same as one in memory, MINC erases the current statement and replaces it with the new one. If you want to save the result of appending files, use SAVE or REPLACE commands.

If you do not have a program in your workspace when you use APPEND, MINC simply gets the file you specify.

When you default the filespec argument, MINC uses the following prompt to ask for the name of the file.

```
OLD FILE NAME—
```

When you are designing separate programs to be merged, choose the ranges of statement numbers carefully. For example, you might use the same statement numbers for all COMMON and DATA statements so that APPEND replaces all but the most recently merged descriptions.

If you are designing a set of small programs to merge as a single program for execution, assign a block of statement numbers to each program. Then the merged program will be correct regardless of the order in which you execute the APPEND commands.

Restrictions

None.

Errors

?MINC-F-Invalid file name

You mistyped the input file name.

?MINC-F-Specified or default volume does not have file named

The file specified in the APPEND command is not on the volume.

?MINC-F-Program too large; workspace overfills

The workspace is too full for the input file.

?MINC-F-Syntax error; cannot translate the statement

The input file specified by the APPEND command is not a BASIC program.

You mistyped the APPEND command.

Related

OVERLAY
CHAIN

References

Book 2, Chapter 14.

Examples

See Book 2, Chapter 14.

Arithmetic

Purpose

MINC provides all the standard arithmetic capabilities required. You can raise a value to a power, multiply, divide, add, and subtract by typing appropriate expressions in either immediate statements or program statements. In addition to those 5 standard operations, MINC has 18 built-in arithmetic functions that perform more complex arithmetic calculations. Calculating sines, finding pseudo-random numbers, using logarithms, and converting strings to numerical values are some of the available arithmetic functions.

Forms

A numeric expression is an expression which yields a single numeric value when evaluated. Numeric literals, numeric variables, functions, and any of these combined with arithmetic operators are all numeric expressions. All of the following are numeric expressions:

```
2
T%
PI
(A^5*((B+5.6)/C))
```

The following table shows the arithmetic operators and the order in which they are applied (their priority in an operator expression).

Operation	*Priority*
()	highest
^	(exponentiation)
* or /	(multiplication and division)
+ or -	lowest (addition and subtraction)

Instructions

Remember the operator priority MINC follows in evaluating numeric expressions and, if necessary, use pairs of parentheses to override the order.

MINC processes the deepest level of parentheses in an expression first; within each set of parentheses, MINC does all exponentiation first, then each multiplication and division from left to right, and finally each addition and subtraction from left to right.

Restrictions

Using parentheses to change the order of evaluation of elements in an expression results in slower evaluation of the expression. If computation speed is important, avoid unnecessary parentheses.

Repeating the same complicated expression slows execution and uses more memory than necessary. When a program uses the result of a complicated expression in more than one statement, calculate the result first, store it in a variable, and use the variable in later statements.

Errors

?MINC-F-Syntax error; cannot translate the statement

Mismatched parentheses or a typographical error.

?MINC-W-Value of integer expression not in range -32,768 to+32,767

?MINC-W-Value of real expression is too small

?MINC-W-Value of real expression is too large

A numeric expression produced a result that MINC cannot store, such as an integer with a value greater than +32,767. Notice that these messages are all warnings (-W-) and not fatal errors. MINC uses the value 0 for each expression that produces one of these messages. Thus, your results are probably wrong whenever you get one of these warning messages, even though your program did not stop running.

?MINC-W-Dividing by zero

An expression resulted in a division by zero. The expression is assigned the value 0 (instead of the mathematically correct value, infinity, which cannot be represented).

Related

Numeric Precision
Routines
See also mixed mode arithmetic in Book 2, Chapter 8.

References

Book 2, Chapters 2 and 8.

Examples

Book 2, Chapters 2 and 8.

Arrays

Purpose

An array is a collection of data stored in the workspace that shares a general name. By using arrays, you can keep track of large amounts of data more easily than by using a large number of separate variables. For example, in MINC a real array resembles a table of real values such as a column of test scores or a multicolumn page of mantissas.

You can use arrays with one dimension or two in your MINC system. An array with one dimension resembles a single column of data; you refer to the different data items by their row numbers. An array with two dimensions resembles a multicolumn table; you refer to the different data items by their row and column indexes.

A virtual array file is like an array, but MINC stores it on a storage volume. A virtual array file can be much larger than the largest workspace array a program can process because MINC keeps only a small part of a virtual array file in the workspace at one time. A virtual array file can be as large as the capacity of the volume. However, MINC processes virtual array files more slowly than workspace arrays.

Almost all arrays require description in a DIM statement. Small arrays with up to 11 rows or 11 columns normally do not require a DIM statement. However, if different programs in a chain are to share a small array, it must be described in a COMMON statement.

Forms

array-name(index)

array-name(row-index,column-index)

Component	*Component Type*	*Component Value*	*Default Condition*
array-name	string or numeric	variable name	required component
row-index	numeric literal	1 to size of workspace	required component
column-index	numeric literal	1 to size of workspace	required component

Instructions

The rules for naming arrays and variables are the same. The name consists of a single letter followed if necessary by a data type designator, % or $ for integer and string arrays respectively. The subscripts for an array are not part of its name but

designate how much workspace is allocated to the array. Therefore, you cannot have a one-dimensional array and a two-dimensional array with the same name.

Arrays and variables are separated in the workspace. Therefore, you can have a variable and an array (either one- or two-dimensional) with the same name. For example, X% and X%(50) are distinct. X% names an integer variable; X%(50) refers to an element in the array named X%. The presence of parentheses (and array indexes) distinguishes variable names from array names.

The following statement describes an array of integers named A.

```
DIM A(12,8)
```

The array A has two dimensions, and the DIM statement describes it as having 13 rows (numbered 0 - 12) and 9 columns (numbered 0 - 8). Thus, array A contains 117 separate elements. Each element contains a single value (in the case of array A, a real value). For example, A(6,6) is one of the elements in array A.

If you refer to an array element without describing the array in a DIM statement, MINC reserves workspace for an array with 11 rows (numbered 0 - 10) and 11 columns (also numbered 0 - 10).

Restrictions

MINC does not reserve workspace for any array until it executes a DIM statement or another statement that uses an element of the array. Therefore, when you suspect that your workspace may be too small for both your program and the arrays you require, describe the arrays explicitly in DIM statements first, use the RUN command to force MINC to reserve space for them, and then continue entering your program statements.

You cannot refer to an element of a virtual array file without describing the file in DIM # and OPEN statements.

For most MINC operations, you refer to array elements rather than to whole arrays. That is, there are no matrix operations in MINC. Some of the MINC data transfer and graphic routines do use whole arrays as arguments.

If an overly large array exceeds the workspace capacity, you can use the CLEAR command to erase all the workspace allocations.

Then modify the DIM statement for the array, use the RUN command to reallocate the workspace and the LENGTH command to check the results of the change.

The lowest-numbered array index is 0. Some people who have programmed in languages where the lowest index is 1 have difficulty remembering to use element 0. You can ignore row 0 and column 0, even though space for them is allocated in the workspace. It is possible to use row 0 and column 0 to store summary information (such as counts or means) about the associated rows and columns. However, using row 0 and column 0 for this purpose increases the chance of undetected errors in index calculations.

Errors

?MINC-F-Array subscript is negative or too large at line XX

A subscript in an array reference has exceeded the bounds of the array. This can happen if you have reversed the order of the subscripts, the subscript is negative, or the subscript has been incorrectly calculated in a loop.

?MINC-F-Array overfills workspace at line XX

The dimension of an array is too large for the array to be held in the workspace.

Related

CLOSE
DIM
INPUT
LINPUT
EXTRA_SPACE
NORMAL_SPACE
OPEN

References

Book 2, Chapter 7.

Examples

None included.

ASC

Purpose

The ASC function takes on the value of the numeric ASCII code corresponding to its single-character argument.

Forms

ASC(character)

Component	*Component Type*	*Component Value*	*Default Condition*
ASC	function name/numeric	0 to 127	required component
character	string expression	any character	required component

Instructions

Use ASC to determine the ASCII numeric code for any character. The character argument for ASC can be one of the ASCII control characters.

Restrictions

The string expression must be exactly one character long.

Errors

?MINC-F-Arguments in definition do not match function called

You gave a non-string expression for the argument, or the string contained more than one character.

Related

**CHR$** CHR$ converts a numeric ASCII code to its corresponding character value. ASC and CHR$ reverse each other's effects. The following identity expresses the relationship:

code = ASC(CHR$(code))

VAL The VAL and STR$ functions are *not* related to ASC. VAL converts a character string to the numeric value corresponding to the string. STR$ converts a numeric value to its corresponding character string representation.

References

Book 2, Chapter 8.
Book 2 appendix, ASCII Character Set.

Examples

Book 2, lower-to-upper-case conversion example (Chapter 8).

Example PRINT ASC('W')
Result 87
Example Enter the CTRL/G combination:
PRINT ASC('^G')
Result 7

Assignment Statement

Purpose

An assignment statement is one of several methods of assigning a value to a variable. With an assignment statement, the presence of an equals sign (=) transfers the value of the expression on the right of the equals sign to the variable on the left of the equals sign.

Forms

LET variable = expression

Component	*Component Type*	*Component Value*	*Default Condition*
LET	statement	none	assigns value
variable	string or numeric variable name	any valid name	required component
=	none	none	required component
value	string or numeric expression	same type as variable	required component

Instructions

Use an assignment statement when you want to change the value of a variable from within a program.

The variable component of the assignment statement is understood to be either a simple variable name (like F) or an array element name (like F(1,10)).

Restrictions

The data type of the variable and the expression must be compatible. It is impossible to assign a string value to a numeric variable (and vice versa). It is possible to assign the results of integer expressions to real variables (and vice versa). The conventions for these assignments appear in Book 2.

Errors

?MINC-F-Invalid operation; mixing numbers and strings

You assigned a string value to a numeric variable or a numeric value to a string variable.

?MINC-W-Value of integer expression not in range -32768 to +32767

?MINC-W-Value of real expression is too small

?MINC-W-Value of real expression is too large

You assigned values to numeric variables that exceed the limits of the variables.

?MINC-F-String is longer than 255 characters

You assigned a string value that has more than 255 characters to a string variable.

Related

With an INPUT or LINPUT statement, the person at the keyboard provides the value for a variable. If you use an INPUT # or LINPUT # statement, the variable assumes a value from a file. If you use a READ statement, the variable takes on a value provided in a DATA statement.

INPUT, INPUT #
LINPUT, LINPUT #
LET
READ and DATA

References

Book 2, Chapters 2 and 3.

Examples

```
10 A=-7
20 B=ABS(A)
30 PRINT A,B
RUNNH

-7             7

READY
```

ATN

Purpose

The ATN function takes on the value of the arc tangent of its argument.

Forms

ATN(expression)

Component	*Component Type*	*Component Value*	*Default Condition*
ATN	function name/real	$-\pi/2$ to $\pi/2$ radians	required component
expression	numeric expression	real range	required component

Instructions

Use ATN to determine the angle whose tangent is the argument. The value of ATN is an angle, expressed in radians, in the range $-\pi/2$ to $\pi/2$.

Restrictions

None included.

Errors

?MINC-W-Value of real expression is too small at line XX

?MINC-F-Arguments in definition do not match function called at line XX

Related

ATN is one of three trigonometric functions available with MINC. The other trigonometric functions are SIN and COS.

References

Book 2, Chapter 2.
Book 3, Numeric Precision.

Examples

Example PRINT ATN(32.345)

Result 1.53989

BIN

Purpose

The BIN function has the numeric value of a string of 1's and 0's representing a binary value.

Forms

BIN(string)

Component	*Component Type*	*Component Value*	*Default Condition*
BIN	function name/integer	-32,768 to +32,767	required component
string	string expression	up to 16 1's and 0's	required component

Instructions

The only argument to the BIN function is a string of 1's, 0's, and optional spaces that represents a binary number. The BIN function ignores the spaces, allowing convenient spacing of the digits. The binary number is treated as a 16-bit signed 2's complement integer, with decimal values in the range -32,768 to +32,767.

Use BIN in the lab module routines to specify digital input and output masks and digital output values. (See Book 6, "Data Types and Number Systems.")

Restrictions

None.

Errors

?MINC-F-Arguments in definition do not match function called

The argument is not a valid binary number.

Related

OCT

References

Book 6.

Examples

```
Example  PRINT BIN('100101001')
Result   297

Example  PRINT BIN('1111 1111 1111 1111')
Result   -1

Example  PRINT BIN('1111 1111')
Result   255

Example  A=BIN('1111 1111'+'1111 1111')
         PRINT A
Result   -1
```

Branching

Purpose

MINC processes program statements in order by statement number. When you have more than one statement on a line, MINC executes the statements from left to right. MINC also provides two techniques to override strictly sequential processing of statements, branching, and loops.

Branches can be either unconditional or conditional. Unconditional branches always cause MINC to transfer control to the statement number you specify in the statement. For example, the following statement transfers control to statement 240:

```
GO TO 240
```

Conditional branches result in transfer of control only if some logical condition you define is true. For example, the following statement transfers control to statement 240 if and only if the value of X is greater than 3.3 when the statement executes.

```
IF X > 3.3 THEN GO TO 240
```

Multiway branches cause MINC to jump to one of several statement numbers in a list. For example, the following statement transfers control to one of the statements (2000, 3000, 4000, or 5000) depending on the value of P. If the value of P were 3, then control would transfer to statement 4000.

```
ON P THEN GO TO 2000,3000,4000,5000
```

Forms

Unconditional branching

The following statements result in unconditional transfer of control:

CHAIN filespec LINE stmt#

GO TO stmt#

GOSUB stmt#

NEXT control variable

RETURN

Conditional branching

IF logical-expression THEN statement

IF logical-expression GO TO stmt#

Multiway branching

ON numeric-expression GO TO ordered-stmt#-list

ON numeric-expression GOSUB ordered-stmt#-list

Instructions

Specific instructions for using the unconditional, conditional, and multiway branching statements appear in the relevant sections of this book.

The following general instructions are offered as guidelines for good programming practices using branching.

- Duplicate short sets of statements where your program requires them rather than using GO TO branches.
- When your program repeats a complicated calculation, consider defining a function to replace the calculation (see DEF).
- When your program repeats a set of statements, consider treating them as a subroutine.
- Include an informative REM statement to describe the purpose of any program branching and the conditions under which it occurs. This investment makes a program readable and is repaid during debugging and modification.

Restrictions

Restrictions about each of the branching statements appear in the separate sections for them.

Errors

The section for each branching statement includes a list of the most common errors occurring with that statement.

Related

FOR
NEXT
CALL
CHAIN

GO TO
ON
IF
GOSUB
RETURN

References

Book 2, Chapters 5, 6, and 7.

Kernighan, B. W. and Plauger, P. J., *The Elements of Programming Style*. New York: McGraw-Hill, 1974.

Examples

See each of the individual sections for examples that apply.

BYE

Purpose

You should use the BYE command immediately after you change the diskette in SY1:. MINC has methods that save time whenever you are performing an operation that uses a diskette. However, if you change the diskette in SY1: without typing the BYE command, these optimizing methods might destroy the files on the newly inserted diskette. Your files will not be destroyed if you type BYE.

The BYE command scratches the workspace just as an SCR command does. The BYE command takes longer than the SCR command, however.

Forms

BYE

Component	*Component Type*	*Component Value*	*Default Condition*
BYE	command	none	required component

Instructions

Always use the BYE command immediately after changing the diskette in SY1:. Note that the BYE command erases the workspace.

Restrictions

Note that whenever you change the system diskette in SY0:, you must execute the RESTART command. See RESTART.

Note that you should not use the SCR command when you change the diskette in SY1:. Although the BYE command scratches the workspace, the SCR command does not do everything that the BYE command does.

Errors

There are no error messages associated with this command.

Related

RESTART
Start Procedures
SCR

References

None.

Examples

Example BYE

Result MINC pauses for a little while and then displays the following message:

```
MINC V1.0

Please enter
Today's date:
```

Calendar Operations

Purpose

You set the MINC system clock and calendar with the TIME and DATE commands. As part of the system start procedure, MINC prompts you to enter the date and time.

Forms

DATE dd-mmm-yy

TIME hh:mm:ss

PRINT DAT$

PRINT CLK$

string-variable=DAT$

string-variable=CLK$

Instructions

Use the DATE and TIME commands to set the date and time. Use the DAT$ and CLK$ functions in immediate or program statements to check the date and time. The sections DATE, TIME, DAT$, and CLK$ have detailed instructions.

For advanced applications, you might be able to use the SCHEDULE, START_TIME, or GET_TIME lab module routines. For example, the GET_TIME routine is more accurate than the CLK$ function for measuring time intervals.

Restrictions

None included.

Errors

See each of the individual calendar commands and functions for the errors.

Related

DATE command
TIME command
DAT$
TIM$

References

Book 2, Chapter 8.
Book 6.

Examples

```
READY
DATE 24-Aug-78

PRINT DAT$
24-AUG-78

READY
```

CHAIN

Purpose

The CHAIN statement automatically connects a series of separate programs or program parts, forming a *program chain*. In each element of the chain, the CHAIN statement is the last statement executed.

The CHAIN statement closes all open files and replaces the program in the workspace with a program from a stored program file. It clears and reallocates the workspace, except for variables and arrays protected by COMMON statements. Finally, it begins executing the element of the chain now in the workspace, starting with either the first statement or the statement specified in the CHAIN statement.

Chaining is the simplest way to prepare separate programs for parts of a complex task and run them without exhausting MINC's workspace.

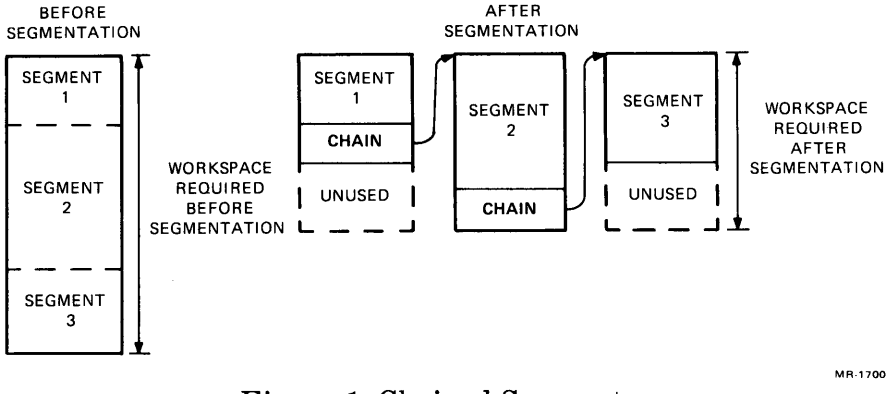

Figure 1. Chained Segments

Forms

CHAIN filespec LINE stmt#

Component	*Component Type*	*Component Value*	*Default Condition*
CHAIN	statement	none	required component
filespec	characters	dev:name.typ	dev: SY0: name NONAME .typ .BAS
LINE	none	none	starts at first statement
stmt#	numeric expression	1 to 32,767	paired with LINE

Instructions

Identify the intermediate values you want each program in the chain to share, and assign variables and arrays for them in COMMON statements to ensure their values are carried from one chained program to the next.

MINC closes all open files when it executes a CHAIN statement. Be sure that each chain element using a sequential or virtual array file contains the necessary OPEN statements for those files.

If you resequence a chain segment, remember to check the LINE argument of the corresponding CHAIN statement.

The CHAIN statement is the last statement executed in each chain element. Although physically there might be statements following the CHAIN statement, none of those statements would ever execute (unless control reached them via a branching statement). Use a REM statement to describe each CHAIN statement to make your program easier to understand.

Save each program in the chain under a different name, and don't forget to save the first program in the chain.

Restrictions

Compiled program files (.BAC files) chain faster than BASIC source programs (.BAS files).

Errors

?MINC-F-Specified or default volume does not have file named

?MINC-F-Program does not have a statement number specified

The line number that you specify in the CHAIN statement does not exist in the program.

?MINC-F-COMMON variables not in the same order as in last program at line XX

Variables or arrays are not specified in the same order in COMMON as in the previous segment.

?MINC-F-Program too large; workspace overfills

The workspace is not large enough to hold the program segment specified by the CHAIN statement.

Related

COMMON The COMMON statements in each element of the program chain protect variables and arrays from being cleared.

OVERLAY The OVERLAY statement merges a segment in a stored program file with the program in the workspace.

APPEND The APPEND command merges a stored program file with the program in the workspace.

References

Book 2, Chapter 14.

Examples

See the file maintenance example in Book 2, Chapter 14. These programs are also included on the Demonstration diskette in files called FILEMT.BAS, FILEM1.BAS, FILEM2.BAS, FILEM3.BAS, FILEM4.BAS, and FILEM5.BAS.

CHR$

Purpose

CHR$ takes on the character value corresponding to the numeric ASCII code of its argument.

Forms

CHR$(code)

Component	*Component Type*	*Component Value*	*Default Condition*
CHR$	function name/string	any ASCII character	required component
code	numeric expression	-32,768 to +32,767	required component

Instructions

Use CHR$ to convert a numeric ASCII code to its corresponding character value. CHR$ is useful for producing the results defined by some of the invisible characters. For example, the line-feed character is invisible and cannot appear in a normal PRINT statement. However, you can produce line feeds with a PRINT statement by specifying the numeric ASCII code for a line feed character (10) as the argument to CHR$.

Several characters are useful for controlling the appearance of screen displays:

Code	*Result*
7	Sounds the terminal's warning tone.
9	Moves to the next horizontal tab stop on the screen (not related to the TAB statement).
10	Moves cursor down one line to produce a blank line equivalent to a single PRINT statement.
13	Returns cursor to start of current line.

Restrictions

The character obtained with CHR$ does not always perform the result defined for that character by the ASCII standard. For example, the codes for CTRL/C (3), vertical tab (11), and form feed (12) have no effect as CHR$ arguments in a PRINT statement.

The numeric ASCII codes shown in the ASCII character table have values ranging from 0 to 127. For any code value outside

that range, CHR$ effectively subtracts (or, in the case of a negative code, adds) 128 from the code value until the value is within the range of ASCII codes.

Errors

?MINC-F-Arguments in function do not match function called

The code argument exceeds the valid range, -32,768 to 32,767.

Related

ASC ASC converts a character to its corresponding numeric ASCII code. ASC and CHR$ reverse each other's effects. The following identity expresses the relationship:

character = CHR$(ASC(character))

**STR$** The STR$ and VAL functions are not related to CHR$. STR$ converts a numeric value to its corresponding character string representation. VAL converts a character string to the numeric value corresponding to the string.

References

Book 2, Chapter 8.
ASCII character code chart, Book 2.

Examples

Lower case to upper case conversion, Book 2.

```
Example  A=CHR$(101)
         PRINT A

Result   e

Example  PRINT CHR$(56)

Result   8

Example  PRINT CHR$(7)

Result   (terminal warning tone sounds)

Example  PRINT CHR$(9);'column heading'

Result           column heading
```

CLEAR

Purpose

The CLEAR command performs a subset of the operations performed by the RUN command.

The CLEAR command has the following effects:

1. Assigns the value 0 to all numeric variables.
2. Assigns the null string to all string variables.
3. Erases all workspace arrays. Any array with a dimension greater than 10 no longer exists.
4. Abnormally closes all open sequential and virtual array file channels.

The CLEAR command does not erase any program statements stored in the workspace.

Forms

CLEAR /

Component	*Component Type*	*Component Value*	*Default Condition*
CLEAR	command	none	required component

Instructions

When MINC displays READY, type CLEAR.

Errors

There are no error messages for this command.

Restrictions

None.

Related

OLD
NEW
RUN The RUN command performs the same action as a CLEAR command before it runs your program.
SCR

References

Book 2, Chapter 2.

Examples

```
READY

10 DIM A(100)
20 FOR I=1 TO 100
30 A(I)=I
40 NEXT I
50 PRINT 'Finished'
RUNNH

Finished

READY
PRINT A(99);A(100)
99 100

READY
CLEAR

READY
PRINT A(99);A(100)

?MINC-F-Array subscript is negative or too large

READY
```

CLK$

Purpose

The CLK$ function takes on the value of the current system time.

Forms

CLK$

Component	*Component Type*	*Component Value*	*Default Condition*
CLK$	function name/string	hours:minutes:seconds	required component

Instructions

Use CLK$ to determine the current system time. The current system time is maintained in 24-hour format so that 5:00 is five o'clock in the morning and 17:00 is five o'clock in the evening. The following table shows the contents of the time string, which takes the form hh:mm:ss.

Position	*Contents*
hh	Hour of the current time; values 0 to 23.
:	Colon
mm	Minute of the current time; values 0 to 59.
:	Colon
ss	Seconds of the current time; values 0 to 59.

You can use the SEG$ and VAL functions to convert CLK$ values to numeric values for calculating elapsed times over long time intervals.

Restrictions

The CLK$ value is inaccurate because the CLK$ function requires time to execute. The expected inaccuracy is about 15 seconds.

Errors

There are no errors associated with this function.

Related

GET_TIME The lab module routine GET_TIME is more precise for measuring elapsed time but requires a clock module.

TIME The TIME command sets the current system time. The CLK$ value corresponds to actual time of day only if the proper time of day was set previously.

References

Book 2, Chapter 8.
Book 6, GET_TIME.

Examples

```
READY
TIME 13:30

READY
PRINT CLK$
13:30:35

READY
```

CLOSE

Purpose

MINC provides a total of 12 channels for data input and data output. MINC assigns them to files on a volume according to OPEN statements in your program. The CLOSE statement saves files associated with channels safely (particularly output files created by your program) and frees the channels for use by other files.

Forms

CLOSE channel-list

Component	*Component Type*	*Component Value*	*Default Condition*
CLOSE	statement	none	required component
channel-list	list of elements	see form	all open channels
Form of list element:			
number-sign	character	#	
channel-number	numeric expression	1 to 12	

Instructions

Complete the CLOSE statement with one or more channel numbers to close specific channels. For example, if the value of A is 15 and the value of B% is 3, the following statements close channel number 3:

```
CLOSE #3
CLOSE (A/5)
CLOSE B%
```

Separate elements in the channel list with commas. MINC accepts real expressions in a CLOSE statement, but when a channel number expression has a fractional value, MINC truncates the fraction. If a channel number expression has a fractional value, MINC truncates the value. For example, each of the following statements closes channel number 6:

```
CLOSE #6
CLOSE 20/3
CLOSE 6.1
```

Restrictions

If a program terminates (either with a STOP statement or abnormally) before its CLOSE statement executes, any open files remain open. The exact state of these open files is unpredictable. In some cases, the complete file is already stored on the diskette. In other cases, portions of the output destined for the file are still in the workspace. If all the file processing is finished when the

program terminates abnormally, you can use a CLOSE statement in the immediate mode to preserve the file.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

The channel expression in the CLOSE statement is invalid.

?MINC-F-Need OPEN statement for file channel at line XX

The channel expression specifies a channel not yet opened.

?MINC-F-File channel is not in the range 1 - 12 at line XX

Related

OPEN
RESET
RESTORE
INPUT
LINPUT
PRINT

References

Book 2, Chapter 11.

Examples

Example	CLOSE #6,# C
Result	Close channel 6 and the channel specified by variable C.

Example	CLOSE
Result	Close all channels previously opened with an OPEN statement.

COLLECT

Purpose

When you erase a file from a volume, the space it filled becomes available for other files or for expansion of the file immediately preceding the space. However, MINC reuses the free space only if it is large enough for a complete file operation. After many file operations, the space on a volume becomes fragmented. That is, a large percentage of its capacity is free but each piece of free space is too small to use.

The COLLECT command puts all files on a volume into consecutive blocks and collects all of the volume's free space. The only exceptions to this process occur if there are bad blocks on a volume. The INITIALIZE command marks the physical locations of bad blocks by creating a file with file type .BAD. These files cannot be relocated by COLLECT. Therefore, small amounts of free space can remain unusable between the .BAD files and the files that precede them.

Forms

COLLECT device

Component	*Component Type*	*Component Value*	*Default Condition*
COLLECT	command	none	required component
device	specification	SY0: or SY1:	required component

Instructions

The best procedure is to use the VERIFY command to check the volume you want to process before collecting its free space. If the volume has been heavily used or handled roughly, a file on it might have developed a bad block. If so, the VERIFY command reports which files now contain bad blocks. If a file contains an unmarked bad block, COLLECT cannot operate properly.

If the volume that you want to collect has unmarked bad blocks in files (other than FILE.BAD), do not collect the volume. See the procedure for recovering these files in the section Error Recovery.

Install the volume you want to process in any available drive and complete the COLLECT command with the drive's device abbreviation. MINC checks the volume owner; if the owner is

DIGITAL, MINC refuses to process any files on the volume and displays a message. Otherwise, MINC processes the volume and signals READY when it finishes.

Restrictions

If the volume has bad blocks that are not marked in the FILE.BAD file, MINC does not display an error message. MINC does collect a volume with unmarked bad blocks, but when you try to use the volume later, you find that the directory does not match the files. Be sure to use the VERIFY command before collecting files.

Errors

?UTILITY-F-Illegal device

The device specified is invalid.

?UTILITY-F-Read error

The device specified is nonexistent.

?UTILITY-F-Error reading directory

There are bad blocks in the directory of the volume being collected.

?UTILITY-F-Uninitialized volume

The volume being collected has never been initialized.

Related

DUPLICATE
VERIFY
INITIALIZE
Error Recovery

References

Book 2, Chapter 4.

Examples

In the following example, 1 bad block was marked in the FILE.BAD file during the initialization. The other bad block is in the user's program file named SINES.BAS.

```
READY

VERIFY
Bad Blocks      Type        Filename        Rel Blk
414             Hard        FILE.BAD        0
417             Hard        SINES.BAS       0
```

To recover the diskette, the user must perform the following steps.

1. Duplicate the diskette using the DUP command.

2. Fix the new copy of SINES.BAS using the keypad editor. (See the Error Recovery section.)

3. Initialize the bad diskette with the INI command to mark the bad block in the FILE.BAD file. Then the user can use the bad diskette again (it is no longer bad).

Comments

Purpose

MINC program statements are easy to read, but the purpose of individual statements and groups of statements in a program can be hard to understand. You are most likely to notice this problem when you try to modify a program written some time ago or by someone else.

You can reduce the problem significantly by taking the time to include explanatory comments in your programs when you write them. Use REM statements for this purpose. The following classes of comments can be useful:

- A banner comment as the first statement in each subroutine, explaining what the subroutine expects from your main program, what the subroutine does, and what it produces for your main program.
- A comment for each function you define in a DEF statement, particularly if the expression that defines the function value is complex.
- A comment for each GOSUB statement, explaining what your main program is providing to the subroutine and what it expects the subroutine to produce.
- A summary comment for each major section of your program.
- A general comment as the first statement in each program, explaining the program's purpose, algorithm, required data and file formats, blocks of statement numbers you have used for different sections, and the procedures people who use the program should follow.
- An explanation of each modification you make to a program after you have debugged it completely.
- An explanation of each variable used in the program.

Forms

REM text

Instructions

The section on the REM statement includes detailed instructions.

REM statements can require a full line, or can fit on the same line as a program statement, separated by a backslash character.

Comments do use workspace. An alternative approach is to develop a separate documentation file for each program and remove most of the general comments that are in REM statements from the program itself. For example, PROTON.BAC and PROTON.DOC, the compiled program and the file of documentation about it, can be complementary.

Restrictions

Do not use a left parenthesis in a remark.

Errors

There are no error messages associated with this statement.

Related

REM

References

Book 2, Chapter 3.

Examples

See Book 2.

COMMON

Purpose

COMMON statements preserve current values of variables and arrays in the workspace for the next element in a program chain. The different programs in a chain share the variable values and arrays that you specify in COMMON statements. The values of all other variables and arrays are strictly local to the specific program in which you use them. MINC erases the workspace when it starts to execute a new chain element, except for the parts of the workspace protected by COMMON statements.

With program chains, putting variables and arrays in COMMON is both faster and simpler than storing intermediate results in a file.

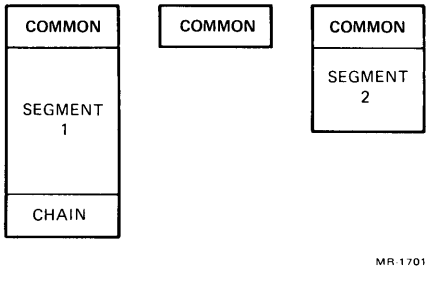

Figure 2. How COMMON Is Stored in the Workspace

Forms

COMMON common-list

Component	*Component Type*	*Component Value*	*Default Condition*
COMMON	statement	none	required component
common-list	list	see form below	required component
Form of list element:			
item	variable name or array description	any valid name	

Instructions

You can use any number of COMMON statements. MINC accepts as many as 255 different variables and arrays in COMMON.

However, whether you use one or several, the variables and arrays you list must be in exactly the same order in the COMMON statements of each of the programs you are chaining together. Separate the variables and arrays you list with commas.

You cannot describe an array in both a DIM statement and a COMMON statement. Remove DIM statements that describe arrays in COMMON. You cannot specify a virtual array file in a COMMON statement because it is a file, not part of the workspace. Use separate OPEN statements in each chaining program that uses a virtual array file.

If a program containing COMMON statements chains to a program that has no COMMON statement, MINC erases all variables and arrays, including those listed in previous COMMON statements. Therefore, it is necessary to include the COMMON statements if you want to use these variables in later chain segments.

It is possible to extend COMMON by placing additional variables and arrays after the previously existing ones; however, you cannot make COMMON smaller.

You can easily make sure that the COMMON statements match for all the segments in a chain by placing the COMMON statements in a separate file and overlaying or appending the COMMON statements to the current segment.

Restrictions

If a program segment containing COMMON statements chains to another program segment containing COMMON, all the variables and arrays in the original COMMON statements should appear in the new statements. But, if the new COMMON statements contain some of the variables and arrays (in the correct order) but not all of them, MINC does not produce an error message. Instead, MINC preserves all the variables specified in the original COMMON statements. Even though no error message is produced, this situation should be avoided because variables are preserved that do not appear in the segment's COMMON statements.

Errors

?MINC-F-COMMON variables not in the same order as in last program

The variables in COMMON are not in the same order in the new segment as in the previous segment.

?MINC-F-Array has invalid description at line XX

An array is dimensioned explicitly with a DIM statement as well as implicitly in a COMMON statement.

?MINC-F-Total number of arrays and variables in COMMON exceeds 255 at line XX

There are more than 255 variables and arrays listed in COMMON statements.

?MINC-F-Program too large; workspace overfills

The new segment is too large for the workspace.

Related

Arrays
APPEND
CHAIN
CLOSE
DIM
OPEN
OVERLAY

References

Book 2, Chapter 14.

Examples

See Book 2, Chapter 14.

COMPILE

Purpose

When you type a program statement, MINC holds it in the workspace in a special form that requires less space than the characters you type. When you list the current program or use the SAVE command to store it, MINC translates each line back to the form you typed. When an OLD command specifies a file you created with the SAVE command, MINC again translates each line back to the special form that conserves space in the workspace.

The COMPILE command stores MINC's special shortened form of the current program. MINC reads compiled programs into the workspace more quickly than uncompiled programs because compiled programs require no translation.

Forms

COMPILE filespec

Component	*Component Type*	*Component Value*	*Default Condition*
COMPILE	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name workspace name .typ .BAC

Instructions

Use the COMPILE command whenever MINC displays READY. When the filespec argument is present, COMPILE stores the workspace program under the name you specify. When the filespec argument is omitted, COMPILE stores the workspace program with the current name of the workspace.

Restrictions

Use the COMPILE command whenever MINC's added speed in reading a compiled program is important to you. Note, however, that a compiled program is less convenient than one you have stored with SAVE or REPLACE in the following cases:

1. An OVERLAY statement cannot refer to a compiled program (a CHAIN statement can do so);

2. The COPY command cannot copy a compiled program to a line printer or to your display (you can copy a compiled program to a storage volume);

3. You cannot use MINC's keypad editor — the EDIT and INSPECT commands — to look at or modify a compiled program;

4. You can use the TYPE command to display a compiled program; however, the characters that appear on the screen are not in a form that you can read.

It is possible to unintentionally destroy a file when you use the filespec argument in the COMPILE command. The argument of the COMPILE command specifies the destination for the compiled version of the workspace, not the program to be compiled. MINC compiles the program currently in the workspace. If you specify an existing (uncompiled) program file in the filespec argument, MINC replaces it with the compiled workspace program. For example, if you use the following sequence of commands, MINC destroys the program in PROG.BAS.

```
READY
SCR

READY
COMPILE PROG.BAS

READY
```

In this sequence, MINC would *not* compile PROG.BAS. Instead, it would compile the program in the workspace (nothing) and put that in the file named PROG.BAS, destroying any previous contents of PROG.BAS. MINC cannot guess your intentions and produces no error message in this case. It is a good idea *never* to compile a program into a .BAS file or any other file with the type other than .BAC.

Errors

?MINC-F-File space allocated on volume is too small

There is not enough room on the volume to put in another file.

Related

CHAIN
OLD
RUN

References

Book 2, Chapter 4.

Examples

See Book 2.

COPY

Purpose

Use the COPY command to copy a file onto another volume, to list the file on a line printer, or to copy a file to a new location on its current volume. The command is a convenient way to make a backup copy of a single file.

Forms

COPY existing-file new-file

COPY filespec LP:

Component	*Component Type*	*Component Value*	*Default Condition*
COPY	command	none	required component
existing-file	characters	dev:name.typ	dev: SY0: name required .typ .BAS
new-file	characters	dev:name.typ	dev:SY0: name existing-file .typ existing-file

Instructions

Use COPY to make a copy of a file with a new name, to copy the file to another volume, or to print the file on your line printer.

MINC checks the owner of the output volume you are using. If the owner is DIGITAL, MINC displays a message and does not complete the command. MINC also checks the file types of both the input file and the output file. If either file is a protected type (.SYS, .COM, .BAD, or .SAV), MINC displays a message and does not complete the command.

When the amount of free space on a volume is critical, you might be able to use the COPY command to gain more contiguous space on a volume after you have used the COLLECT command. Check for free space preceding a .BAD file that is large enough for a file that is listed later in the directory. COPY the file from the current volume to the same volume. MINC will move the file forward in the volume's directory to the free space that COLLECT could not use. Finally, process the volume again with COLLECT.

Restrictions

Note that the COPY command cannot copy a file that has bad blocks. Thus, to recover a file that contains bad blocks, you must duplicate the entire volume with the DUP command and then edit the new copy with the keypad editor. (See the Error Recovery section of this manual.)

Errors

?UTILITY-F-File not found

The file that you are copying from does not exist.

?MINC-F-Output file name is already in use;
do you want to erase its current contents (Y or N)?

If you want to destroy the old file, type Y; otherwise type N.

?MINC-F-COP requires two filenames

You forgot to type one of the two file specifications required by the COPY command.

?UTILITY-F-Read Error

The file you are copying from has bad blocks.

?UTILITY-F-Write Error

The file you are copying to has bad blocks.

?UTILITY-F-Not enough usable space on volume

There is no available space on the volume for a copy of the file.

?UTILITY-F-Volume owner may not be 'DIGITAL'

DIGITAL is the owner of the destination volume. (You made a mistake and placed a Master diskette in SY1:.)

?MON-F-Directory I/O error

There are bad blocks in the directory of the volume being copied to.

Related

DUPLICATE
COLLECT
TYPE
LIST
PRINT

References

Book 2, Chapter 4.

Examples

See Book 2.

COS

Purpose

The COS function takes on the cosine of its argument.

Forms

COS(expression)

Component	*Component Type*	*Component Value*	*Default Condition*
COS	function name/real	-1 to +1	required component
expression	numeric expression	any value, radians	required component

Instructions

Use COS to determine the cosine of any angle.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

COS is one of three trigonometric functions provided by MINC. The other trigonometric functions are SIN and ATN.

References

Book 2, Chapters 2 and 8.

Examples

```
Example   F=COS(PI)
          PRINT F

Result    -1

Example   PRINT COS(PI/2)

Result      0

Example   PRINT COS(10*PI)

Result      1
```

CREATE

Purpose

The CREATE command invokes the keypad editor so that you can create a file. With the keypad editor, you can create a sequential ASCII file that contains a BASIC program, text, or any other characters you want.

Forms

CREATE filespec

Component	*Component Type*	*Component Value*	*Default Condition*
CREATE	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name required .typ .BAS

Instructions

Use the CREATE command to invoke the keypad editor so that you can create a file.

The filespec represents the name of the file that you are creating.

After you execute the CREATE command, the keypad editor erases the screen, leaving only the cursor in the left-hand upper corner. To put characters into this file, simply type the characters.

Restrictions

It is possible to create a program file using the editor that the OLD command cannot load into the workspace. Programs must not contain completely blank lines or statement numbers with no statement following them. If you have problems bringing a program into the workspace, use EDIT to find the cause of the problem and fix it.

Errors

?EDITOR-W-Limited space for insertions (only 0 blocks)
?EDITOR-W-Continue (Y or N)?
?EDITOR-F-Output volume has maximum number of files or no free blocks

Related

EDIT The EDIT command invokes the keypad editor. You can modify an existing file or create a new file which is a modified version of an existing file.

References

Book 2, Chapter 15.

Examples

None included.

CTRLC

Purpose

The CTRLC system function reenables the normal control operation of the CTRL/C combination (disabled by RCTRLC). After the CTRLC function executes, you can terminate program execution by entering the CTRL/C combination.

Forms

variable=CTRLC

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric	unknown	required component
=	none	none	required component
CTRLC	function name	none	required component

Instructions

Use CTRLC to reenable the operation of CTRL/C after it has been disabled by RCTRLC. The value of the variable is unknown after the function has executed. You do not need to use the variable; it is part of the syntactic form of the function only.

Restrictions

It might sometimes be wise to disable CTRL/C during crucial program runs. However, if CTRL/C is disabled, it is impossible to stop the program and preserve the workspace if problems develop.

The CTRLC function applies to programs only. The utility that produces the READY message enables CTRL/C. Therefore, either normal or abnormal program termination reenables CTRL/C.

Errors

There are no error messages associated with this function.

Related

RCTRLC The RCTRLC function disables normal CTRL/C operation.

References

Book 6, PAUSE.

Examples

None included.

CTRL operations

Purpose

Computer industry standards have established 33 ASCII character codes for special technical applications. Most of these stand for characters you type by holding down the CTRL key and pressing another key.

Some of these control characters have special meanings to MINC. They are listed below. The other control characters are not useful to MINC.

Forms

CTRL/key

Instructions

CTRL/C MINC immediately stops whatever process it is performing and displays READY. If the program is waiting for keyboard input, one CTRL/C is sufficient. Otherwise, enter two CTRL/C characters to stop the program.

CTRL/O Any text output to the screen is suspended. The output continues internally to MINC but does not show on the screen. Another CTRL/O reenables the screen display, and output continues at the current character (*not at the character where it originally halted*). CTRL/O affects the terminal screen only.

CTRL/S, CTRL/Q These characters suspend and reenable screen displays. Their operation is the same as that provided by the NO SCROLL key. CTRL/S suspends screen display. CTRL/Q reenables screen display and output continues with the character following the last one appearing on the screen.

CTRL/U This character deletes all characters entered since the most recent RETURN key. In the keypad editor, CTRL/U deletes the characters from just left of the current cursor position to the beginning of the line.

CTRL/W The CTRL/W character causes the keypad editor to redisplay the current screen. Normally, you would use this character if you used SETUP to change the screen width while you were using the keypad editor. The CTRL/W character causes MINC to redisplay the screen using the new width.

Restrictions None included.

Errors These keys cause no error messages to appear.

Related
SYS system function
7-bit ASCII code
CTRLC, RCTRLC, and RCTRLO system functions

References Book 2, Chapter 1.

Examples None included.

DAT$

Purpose

The DAT$ function takes on the value of the current system date.

Forms

DAT$

Component	*Component Type*	*Component Value*	*Default Condition*
DAT$	function name/string	day-month-year	required component

Instructions

Use DAT$ to obtain the current system date. Some experiments require keeping date records as part of the data.

The DAT$ value is a nine-character string. The following characters comprise the string, which takes the form dd-mmm-yy.

Position	*Contents*
dd	Day of the month, 1 through 31
-	Hyphen
mmm	Abbreviation for the English name of the month (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC)
-	Hyphen
yy	Last two digits of the current year

Restrictions

If MINC has been running overnight at the end of a month, the system's current date is not necessarily correct. Set the date once a day if the correct date is necessary for your application. The startup procedure always prompts you to enter the current date. It is useful to enter the correct date since the date is stored with files that are created or modified.

Errors

No error messages are associated with this function.

Related

DATE is the command that sets the current system date. You can also use DATE to display the current date on the screen.

References

Book 2, Chapter 8.

Examples

```
READY
DATE 8-Aug-79

READY
PRINT DAT$
08-AUG-79

READY
```

DATA

Purpose

A DATA statement provides values to read with a READ statement.

The best use of DATA statements with READ statements is to use them to assign values to variables that do not change within one run of a program but might change from run to run. In this way, all pertinent values are in one place and can be changed easily between runs. See Chapter 12 of Book 2 for a good example of this.

Forms

DATA data-list

Component	*Component Type*	*Component Value*	*Default Condition*
DATA	statement	none	required component
data-list	list	see form below	required component
Form of list element:			
value	string or numeric	type corresponds to READ variables	required component

Instructions

Use DATA statements to provide values for READ statements.

Note that MINC reads values in DATA statements in statement number order, from left to right within each DATA statement. RESTORE (and RESET) cause MINC to read the next value from the first DATA statement in the program.

Separate the values in DATA statements with commas. If string values must contain commas, enclose these strings in quotes. Values must be in the order expected by the READ statements. The data type of each value (numeric or string) must match the data type of the corresponding READ variable.

The DATA statement must be the last statement in a line because MINC processes everything between the word DATA and the next RETURN key as data values.

Restrictions

If a string variable in a READ statement corresponds to a numeric value in a DATA statement, MINC assigns the ASCII character representation of the number to the string variable, not the numeric value.

A DATA statement cannot have any statements following it on the line.

Errors

?MINC-F-DATA value or value from file does not match variable at line XX

The READ statement is expecting a numeric variable, but the corresponding value in the DATA statement is a string.

The DATA statement is not the last statement in a multiple statement line (which causes MINC to interpret the last value as a string value).

?MINC-F-Too few values for INPUT or READ variables at line XX

The DATA statement does not have enough values to correspond to the number of variables in the READ statement.

The DATA statement is missing.

Related

Assignment You can assign values to variables directly in an assignment statement, as in the following example.

```
A=5
```

INPUT # You can use separate files of data you have stored under unique names. Open a file with an OPEN statement, and then read the values with INPUT # or LINPUT # statements, as in the following short example. (The example assumes that the file SY0:VALUES.DAT has integers as the first three values.)

```
OPEN values FOR INPUT AS FILE 6
INPUT 6, A%, B%, C%
```

INPUT You can type each value interactively as the program needs it as in the following general example.

```
PRINT 'Enter an order number —'
INPUT I9$
```

READ

RESTORE/RESET

References

Book 2, Chapter 12.

Examples

Book 2, Chapter 12.

The lab module routines SCHEDULE and PAUSE provide a mechanism for scheduling events to occur at specific times of day.

Errors

?KMON-W-Illegal date

The date you entered is in an incorrect format.

Related

TIME
DAT$
CLK$
LIST
RUN
Calendar functions
DIRECTORY
SCHEDULE
PAUSE

References

Book 2, Chapter 8.

Examples

```
READY
DATE 3-JAN-79
```

DEF

Purpose

MINC includes 31 functions that provide common but somewhat complex calculations that you can use by referring to them by name. For example, the LOG10 function takes as its value the base-10 logarithm of a numeric value, and the LEN function takes as its value the number of characters currently in a string variable.

You can define similar functions in DEF statements. For example, if you use tangent values often, defining your own tangent function once may both save programming time and shorten your programs.

Other statements can use that tangent function without writing out the calculation fully.

Forms

DEF FNx-symbol(dummy-argument-list)=expression

The following forms correspond to the different values for the symbol argument.

DEF FNx(dummy-argument-list)=numeric-expression

DEF FNx%(dummy-argument-list)=numeric-expression

DEF FNx$(dummy-argument-list)=string-expression

Component	*Component Type*	*Component Value*	*Default Condition*
DEF	statement	none	required component
FN	function name	none	required component
x	function name/letter	A to Z	required component
symbol	character	null,%,$	required component
dummy-argument- list	list	variable name(s)	required component
expression	real,integer,or string	type matches symbol	required component

Instructions

Functions you define must have names that begin with FN. Choose any letter as the third letter of the function name. Add a percent sign (%) if you want the function to take on an integer value. Add a dollar sign ($) if the function takes on a string value. No data type symbol is required as part of the name if the function takes on a real value.

Complete the DEF statement with a list of dummy argument variables and a defining expression. The defining expression uses any literals and other functions that are appropriate and can (but need not) use the dummy variables. The defining expression can include another user-defined function but cannot reference itself.

Each function you define must be complete on one line.

Restrictions

The DEF function operates in programs only. When you try to use DEF in immediate mode, it fails but no message appears.

Any functions defined in a program remain valid definitions after the program finishes executing. Therefore, you can use a user-defined function in immediate mode if it was defined in a previous program (and the workspace has not been erased).

Note that a function remains defined after an overlay as long as the DEF statement exists in the workspace.

Errors

?MINC-F-No DEF statement for the function named

You are referring to a function that you tried to define in immediate mode.

?MINC-F-An earlier statement already defined the function at line XX

?MINC-F-Invalid operation; mixing numbers and strings at line XX

The type of the expression (string or numeric) does not match the type of the function.

Related

Naming

References

Book 2, Chapter 8.

Examples

See Book 2.

DEL

Purpose

Frequently you need to remove selected statements from programs. The DEL command erases statements in the current program according to the individual statement numbers and statement number groups you specify.

Forms

DEL firstline - lastline, firstline - lastline, ...

Component	*Component Type*	*Component Value*	*Default Condition*
DEL	command	none	required component
firstline	numeric literal	1 to 32,767	first program statement
-	character	-	deletes one statement
lastline	numeric literal	1 to 32,767	last program statement

Instructions

Enter the command when MINC displays READY. The numeric order of items in the list is not important. Separate individual statement numbers and groups with commas. When you specify a group of statement numbers, the first-line argument must be less than or equal to the last-line argument. For example, 150-200 is a correct group and 200-150 is an incorrect group.

Remember that when you want to erase a single statement, you can simply type its number followed by a RETURN.

Restrictions

Neither of the statement numbers has to exist in the program. However, if you specify an invalid DEL command, MINC does nothing, but it gives you no message.

Once you have deleted a statement, it is irretrievably gone. For large deletions, it might be useful to use the keypad editor so you can see which lines are being deleted.

Errors

The DEL command does not generate any error messages. However, the following erroneous conditions do not cause unexpected deletions.

If none of the statement numbers in a given range exist, nothing happens.

If the statement numbers in the list form an invalid argument,

MINC does not delete those statements. For example, the following statement is incorrect, and DEL does not delete any statements or display an error message.

```
DEL 50-20
```

Related

Retyping a statement
Multiple statement lines
Erasing

References

Book 2, Chapter 4.

Examples

Example DEL 10

Result Deletes statement 10

Example DEL 10-20

Result Deletes statements 10 through 20, starting with the first statement number that is 10 or more and ending with the last number that is 20 or less.

Example DEL 100-150,15,200,50-65

Result Deletes statement 15, the group from 50 through 65, the group from 100 through 150, and statement 200.

Example DEL 2640-2640

Result Deletes statement 2640

Example DEL 100-

Result Deletes all statements numbered 100 or higher.

Example DEL -250

DEL 0-250

Result Both commands delete all statements from the beginning of the program to statement 250 (inclusive).

DIM

Purpose

MINC automatically provides workspace for arrays with no more than 11 rows or columns. This space is not allocated, however, unless you make reference to such an array. Use DIM statements to describe larger arrays and virtual array files and larger arrays. Arrays are described only once. (When you describe an array in a COMMON statement, you must not also describe it in a DIM statement.)

Forms

DIM # channel, arrayname(high-row-index,high-column-index)=length

Component	*Component Type*	*Component Value*	*Default Condition*
DIM	statement	none	required component
#	character	#	paired with channel
channel	numeric literal	1 to 12	virtual array file
array name	characters	valid array name	required component
high-row-index	numeric literal	1 to workspace size	required component
high-column-index	numeric literal	1 to workspace size	single dimension array
=	statement component	none	paired with length
length	numeric literal	1 to 255	16

If the #channel argument is present, then the DIM statement defines a virtual array file. The channel number remains associated with the virtual array file until the file is closed.

The length argument defines the number of characters in each element of a string virtual array file. All elements in a string virtual array file have the same length. To conserve space on a volume, keep the length argument as small as possible.

Instructions

You can describe more than one array in a single DIM statement. However, you cannot describe more than one virtual array file in a DIM # statement. Choose an array name according to the same rules as for variables. Describe the size of the array in terms of its highest row number (and highest column number, if it has two-dimensions). For example, DIM A9$(25,25) describes the string array A9$ as a two-dimensional array with rows 0 through 25 and columns 0 through 25.

MINC requires a channel number as part of each virtual array

file description. For example, DIM #9, B9$(25,25) is a virtual string array accessible on channel number 9. B9$ also has rows 0 through 25 and columns 0 through 25.

MINC accepts DIM statements anywhere in a program. It is often easier to modify programs if all the DIM statements are collected in one area of the program.

Restrictions

The DIM statement applies only to program mode. You cannot describe an array in immediate mode. You can try it, but it fails without a message.

Large arrays require large amounts of workspace. If adding arrays makes one of your programs too large for the workspace, consider the following alternatives.

- Use virtual array files instead of large arrays. Programs using virtual array files execute more slowly, but the virtual array file requires workspace equivalent to a 256-element integer array.
- Use temporary sequential data files instead of large arrays.
- Segment your program into smaller independent units that can execute sequentially. Use COMMON and CHAIN statements to define the order for executing the units and save each one as a separate file. The CHAIN statement clears the workspace between chain units, deleting arrays that are no longer necessary.
- Use OVERLAY statements to merge a main program with different subroutines at different times. In this way, different segments of the program share the same arrays but the program itself does not consume much workspace.
- You cannot conditionally dimension an array. For example, the following statement does not work.

```
10 IF A<0 THEN DIM B(100)
```

Errors

?MINC-F-Syntax error; cannot translate the statement at line XX

Besides looking for a typographical error, you can get this message if you have dimensioned an array with more than two subscripts.

?MINC-F-Array overfills workspace at line XX

The dimension statement tries to create an array that is larger than the workspace can hold.

?MINC-F-Array has invalid description at line XX

The DIM statement is syntactically correct, but invalid. For example, the dimension is a negative number or a variable.

An array is described in both a DIM statement and a COMMON statement.

Related

Arrays
COMMON
CHAIN
OVERLAY

References

Book 2, Chapters 7 and 11.

Examples

Example	DIM #1, F$(250,6)=20
Result	Describes a string virtual array file where each element in the string array is 20 characters long.

DIRECTORY

Purpose

MINC users frequently need to determine which files are stored on the volume they are using and how much unused space remains. The DIRECTORY command causes MINC to display a list of the files on a volume. The list includes each unused area on the volume, the volume's identifier and owner, the creation date for each file, and the size of each file.

Forms

DIRECTORY filespec outputspec

Component	*Component Type*	*Component Value*	*Default Condition*
DIRECTORY	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name all files .typ all files
outputspec	characters	dev:name.typ or LP:	dev: SY0: name required .typ .DIR

Instructions

By defaulting components of the command, you can specify how much information about a diskette you want MINC to report. The four types of reports you can request are as follows:

1. A directory of all files on a diskette (DIR dev:).

2. A directory of all file names on a diskette that have the specific file type you request (DIR dev:.typ).

3. A directory of all file types on a diskette that have the specific file name you request (DIR dev:name).

4. The directory entry for the single file you specify on a diskette (DIR dev:name.typ).

If you specify an outputspec argument, MINC routes the directory report to that file. If your MINC system has a line printer, you can specify LP: as the output device and MINC prints the directory report on it.

Use any form of the command when MINC displays READY. Unless you specify an output file, MINC displays the directory you request on your terminal.

Directory listings for system volumes and for volumes that do not contain a system show one important difference: the amount of space available. A system volume contains the MINC system files. Although these do not appear in the directory listing, they do use space on the volume. Nonsystem volumes contain only the programs and data files you have created. Thus on a nonsystem volume, the sum of the file sizes plus the total free space is 960 blocks.

For a system volume, the sum of the block sizes and free blocks in the directory listing is not equal to the total capacity of the volume.

If you ask for a directory listing of a volume owned by DIGITAL, MINC displays the volume identifier and owner, and it will list a small number of files (depending on which DIGITAL diskette you are using). Note that you cannot write any data to a volume DIGITAL owns.

Restrictions

None included.

Errors

?MINC-F-Invalid file name(s)

?DIR-F-Error reading directory

There are bad blocks in the directory.

?DIR-F-Illegal directory

The diskette is uninitialized.

Related

OLD
LIST
LISTNH
Protected file types
UNSAVE
KILL
NAME
RENAME
REPLACE
INITIALIZE
Line Printer

References

Book 2, Chapter 4.

Examples

Example	DIR
Result	Gives a directory of SY0:
Example	DIR SY1:
Result	Gives a directory of SY1:
Example	DIR .BAS
Result	Gives a directory of all the files with type .BAS on SY0:.
Example	DIR SY1:.BAS
Result	Gives a directory of all the files with type .BAS on SY1:.
Example	DIR NAME
Result	Gives a directory of all the files with filename NAME.
Example	DIR SY1:NAME
Result	Gives a directory of all the files with name NAME on SY1:.
Example	DIR NAME.DAT
Result	Shows the file NAME.DAT if it is on SY0:.
Example	DIR SY0: LIST
Result	This example creates a file on SY0: called LIST.DIR and sends the directory listing to the file instead of to the terminal screen.

DUPLICATE

Purpose

The DUPLICATE command is the most convenient way to copy the entire contents of a diskette to a new diskette. MINC also accepts the abbreviation DUP. The command copies all files from the diskette in device SY0: to the diskette in device SY1:. For a system volume, DUPLICATE copies the system utility files as well as your files.

The DUPLICATE command can duplicate a file with bad blocks. Thus, if one of your diskettes develops a bad block in one of your programs, you can partially rescue the file. However, you have to fix the file on the new diskette with the keypad editor (see Keypad Editor). Note that the COPY command will not copy a file with bad blocks.

If the bad blocks are in the MINC system programs, you can duplicate the diskette to save your own files, but do not use the diskette as a system diskette. The new diskette has no physical bad blocks, but the problems with the system files are not fixed. You cannot use the editor to fix the system programs. To create a new valid system diskette, you must duplicate a working system diskette.

Forms

DUPLICATE

Component	*Component Type*	*Component Value*	*Default Condition*
DUPLICATE	command	none	required component

Instructions

Use INITIALIZE to prepare a new diskette before typing DUPLICATE. Use DUPLICATE to create backup copies of important system, program, or data diskettes. When you enter the DUPLICATE command, MINC prompts you to insert an initialized (and empty) diskette in your SY1: drive and the diskette you want to copy in SY0:. Note that you do not have to put a system diskette in SY0: at this time. When you have inserted both diskettes correctly, press the RETURN key. When MINC finishes the DUPLICATE processing, it asks you to insert your system diskette in SY0: and press the RETURN key.

The following systematic procedures are recommended for creating volume duplicates.

- Use the VERIFY command to check for bad blocks on

the diskette you want to duplicate. If a file has developed bad blocks, you must remember to fix it after you have duplicated it to a new diskette. Remember, if there is a bad block in one of the system files, you cannot use the new copy as a system diskette.

- Write up a new paper label for the duplicate diskette you want to make. Try to avoid writing on the label after you paste it on the diskette.
- Finally, use the DUPLICATE command to make the duplicate copy. When MINC completes it, you can also list and attach the copy's current directory, if your system has a printer.

Restrictions

DUPLICATE can encounter problems if the diskette receiving the duplicate has bad blocks. That is, if you insert a diskette with bad blocks in SY1:, the duplication procedure might not work, depending on where the bad blocks actually occur on the diskette. If the procedure does not work, MINC displays the following error message.

?UTILITY-F-Device full

In this case, you cannot use this diskette as a system diskette. You can, however, use this diskette to store programs or data. Because the diskette has been initialized, all of the bad blocks are marked in the file called FILE.BAD and thus do not affect the storage of programs or data.

Errors

?UTILITY-F-Target volume must be newly initialized

The volume in SY1: is not initialized.

The volume in SY1: is not empty; that is, it contains valid files. You must initialize it before you duplicate.

?UTILITY-W-No volume id

Volume in SY0: is not initialized.

?UTILITY-F-Illegal directory

Volume in SY0: is not initialized.

?UTILITY-F-Device full

This message might appear if you have a bad block in the

directory or if a bad block occurs in such a place as to prohibit a system file from fitting into the available space.

Related

COPY
INITIALIZE
COLLECT
Error Recovery

References

Book 2, Chapter 4.

Examples

The following example shows the entire procedure for duplicating a diskette.

```
INI SY1:
Install volume to be initialized in SY1, and press RETURN (RET)

Current volume id:eeeeeeeeeeee
Current owner: eeeeeeeeeeee
Proceed with initialization (Y or N)?Y
Type new Volume id:MINC system
Type new owner name:Student 003

Initialization is complete; found 000 bad blocks

READY
verify sy1:
There were no bad blocks found

READY
DUP
Install volume to be duplicated in SY0;

install initialized, empty volume in SY1, are you ready (Y or N)?Y

SY1 volume id is: MINC system
SY1 owner is: Student 003

Do you want to duplicate another volume (Y or N)?N

Re-install system volume in SY0, and then press RETURN (RET)

MINC V1.0

10-MAY-78
04:33:40

READY
```

EDIT

Purpose

The keypad editor allows you to add to, delete from, and change ASCII files. You can edit BASIC programs with the editor as well as with BASIC. However, the editor is the only way to edit sequential ASCII files that are not programs.

The EDIT command invokes the keypad editor. EDI is the valid abbreviation.

Forms

EDIT input-filespec output-filespec

Component	*Component Type*	*Component Value*	*Default Condition*
EDIT	command	none	required component
input-filespec	characters	dev:name.typ	dev: SY0: name required .typ .BAS
output-filespec	characters	dev:name.typ	dev: same as input name same as input .typ .BAS

Instructions

Use the EDIT command to begin editing an existing ASCII file. To create a new file, use the CREATE command. To look at a file without changing it, use the INSPECT command.

When you specify the EDIT command with only an input file specification, the editor creates a temporary file that holds the input file unchanged. When you store the file by pressing the STORE FILE key, the editor renames the original file dev:filename.BAK and creates a new file with your input file specification. With repeated editing of the same file, EDIT replaces the previous version of dev:filename.BAK each time.

When you specify the EDIT command with two different file names, the editor stores the file with your edits in the second file specified when you press the STORE FILE key. In this case, the editor does not alter the input file specified. This provides a method for creating a new file which is a modified version of an existing file while avoiding typing the whole file again.

If you specify the EDIT command with the same file name twice, when you press the STORE FILE key, the changed file is in the file name specified and the previous version of the file is erased.

The default conditions for EDIT filespec arguments are relatively complex.

1. If you specify the volume for the output file, you must also specify the file name.

2. If you specify any part of the output file specification, the editor defaults the extension to .BAS unless you specify the extension.

Adding text to a file increases file size quickly. (One file block can contain at most 512 text characters.) The EDIT command protects you in the following ways against adding too many characters and not being able to store the file when you are finished.

1. When you enter the EDIT command, the volume must have a free space at least as large as the current size of the file. If there is not a large enough free space, EDIT does not permit you to edit the file.

2. The maximum size of a file that you can create with the editor is 480 blocks. The editor does not insert any characters into a file once the file size reaches the maximum.

3. When a file has reached its maximum size, the terminal warning tone sounds every time you try to input a character. No error message appears, and the characters do not appear on the screen.

Restrictions

You can modify programs with EDIT in such a way that the OLD command cannot bring them into the workspace. Remember that programs cannot contain empty lines and statement numbers followed by a blank line. The keypad editor does not monitor program syntax.

You should never edit ASCII virtual array files or any non-ASCII files such as numeric virtual array files or .BAC files. These file types should be manipulated only by BASIC.

Use the DISPLAY_CLEAR graphic routine to erase graphics that appear on the screen before using the EDIT command.

Some of the control characters have different meanings to the editor than they do to MINC. These control characters are as follows.

CTRL/O The CTRL/O character stops all output and input to the terminal. A second CTRL/O character resets the terminal.

Note that when you press the CTRL/O character the first time, you might get a spurious character on the screen. However, this character goes away when you reset CTRL/O.

When you reset CTRL/O, the cursor is not always in the same place on the screen as MINC thinks it is internally. CTRL/W refreshes the screen so that the cursor is in the same place on the screen as it is internally.

CTRL/S, CTRL/Q These characters work the same in the editor as in BASIC.

CTRL/U The CTRL/U character deletes from the current cursor position to the beginning of the line. Note that this operation is different than the CTRL/U operation in BASIC.

Errors

?EDITOR-W-Limited space for insertions (only 0 blocks)

?EDITOR-W-Continue (Y or N)?

?EDITOR-F-Output volume has maximum number of files or no free blocks

?EDITOR-F-No output space large enough to EDI input file

?EDITOR-F-Unable to size screen

This message occurs when you type ahead after terminating an EDIT session by pressing the STORE FILE key.

Related

CREATE Use the CREATE command to create a new file by entering text from the keyboard.

INSPECT Use the INSPECT command to display the contents of a file without changing it.

File Allocation

Keypad Editor This section discusses the MINC keypad editor in general.

References

Book 2, Chapter 15.

Examples

See Book 2, Chapter 15.

END

Purpose

Always include an END statement as the highest numbered statement in your programs. END causes MINC to terminate all program processing, close all files that are open, and display the READY signal.

Forms

END

Component	*Component Type*	*Component Value*	*Default Condition*
END	statement	none	required component

Instructions

A program can have only one END statement, and its statement number must be the highest statement number in the program. MINC does not give an explicit message when it executes an END statement; it simply displays READY.

It is a good idea always to give the END statement the number 32767 (the largest statement number).

Restrictions

None

Errors

?MINC-F-END statement does not have highest number in program at line XX

The END statement is not on the last line of the program.

The END statement is not the last statement on a multiple statement line.

Related

STOP The STOP statement can be placed anywhere in the program.

References

Book 2, Chapter 3.

Examples

See Book 2.

Erasing

Purpose

Erasing files, lines, characters, volumes, and parts of memory is a fundamental operation in using MINC. Each of the different MINC commands and statements that erase something has a different purpose and a particular set of operating rules. For detailed information read the appropriate section.

Forms

APPEND filespec
CHAIN filespec LINE stmt#
CLEAR
CTRL/U
DEL first-line - last-line
DELETE key
INITIALIZE SY1:
KILL 'filespec'
NEW filespec
OLD filespec
OVERLAY 'filespec' LINE stmt#
REPLACE filespec
SCR
SUB stmt# [current-form[changed-form[occurrences
UNSAVE filespec

Instructions

Commands

APPEND	Erases each line in the current program that has the same statement number as an incoming statement. Use APPEND when MINC displays READY to accomplish the same effect as an OVERLAY program statement.
CLEAR	Erases the contents of all variables and arrays in the current program. The command does not erase any program statements. By erasing the current arrays and variables, the command reduces the size of the program in the workspace until you run the program again.
CTRL/U	Internally erases all characters on the line you are typing. The line still appears on your terminal screen, however.
DEL	Erases statements you specify from the current program.

DELETE key	The DELETE key erases the character just to the left of the cursor. It applies only to the current line (except in the keypad editor).
INITIALIZE	Erases all files from the volume in the device you specify.
NEW	Erases the current program from memory (but the command does not affect any copy of the program on a volume).
OLD	Erases the current program from memory and loads the workspace with the file you specify.
REPLACE	Erases the current contents of the file you specify and replaces them with the program currently in the workspace.
SCR	Erases the current program and changes the current workspace name to NONAME.
SUB	Erases a string in a current program statement and substitutes a string you specify in its place.
UNSAVE	Erases the specific file you name from the device you specify.

Statements

CHAIN	Erases all variables and arrays except the ones that are listed in both the current program's COMMON statement and the incoming program's COMMON statement. COMMON preserves variables and arrays in chains.
KILL	Erases the specific file you name from the device you specify. A KILL statement in a program has the same effect as an UNSAVE statement when MINC displays READY.
OVERLAY	Erases each statement in the current program that has the same statement number as an incoming program statement. The incoming statement replaces the current statement.

Restrictions

See each of the individual sections for the restrictions that apply.

Errors

See each of the individual sections for the errors that apply.

Related

The section on the keypad editor describes each of the editor's erasing functions.

The RENAME command and the NAME statement change the name of a file in a volume's directory, but they do not change the file itself in any way.

References

Book 2.

Examples

See the individual sections.

Error Recovery

Purpose

This section describes how to recover from the following error conditions.

- Bad blocks that develop in one of your files.
- Bad blocks that develop in the MINC system files.
- The @ character that occurs when you are typing or when you accidentally press the BREAK key.

Forms

Not applicable.

Instructions

Bad blocks occur in your files

The procedure for recovering a file with bad blocks is as follows.

1. Use the VERIFY command on the diskette with the problem file(s) to determine which files are bad.

2. Use the INI command to initialize a new diskette.

3. Use the DUP command to duplicate from the bad diskette to the new one. Note that on the new diskette, these files show no bad blocks because physically the diskette is good. However, the files are not correct because they were damaged when the old diskette was damaged.

4. Use the keypad editor to fix those files that had bad blocks. When you edit the files that previously had bad blocks, you might see erroneous characters that you can fix with the editor.

5. You can use the bad diskette again after using the INI command to reinitialize it and mark its bad blocks with FILE.BAD

Note that you cannot fix virtual array files or compiled programs (.BAC extension).

Bad blocks occur in the system files

The procedure for recovering your files from a diskette with bad blocks in the system files is as follows.

1. Place a working system diskette in SY0:.

2. Type the DUP command. When the DUP command prompts you, place the bad system diskette in SY0: and a new, initialized diskette in SY1:, and proceed with the duplicate procedure.

3. When the DUP command prompts you to put a system diskette in SY0:, be sure to put a working system diskette in SY0:. Neither your bad diskette nor the copy you made of it are working system diskettes.

4. Use the keypad editor to restore any of your own files that had bad blocks. Be sure not to use the copy as a system diskette because you cannot use the editor to recover the ruined system files.

@ occurs when you are typing

The @ occurs when you have pressed the BREAK key.

1. Type P (note, the P must be a capital P) followed by a RETURN. This command should cause MINC to resume processing where you left off. When you use P after an @ occurred, MINC assumes you typed a RETURN at the end of the line you last typed.

2. If P does not work, type 173000G (capital G). This command restarts MINC. Thus, your workspace is scratched.

Restrictions None included.

Errors None included.

Related

DUPLICATE
VERIFY
RESTART

References None included.

Examples None included.

EXP

Purpose

The EXP function takes on the value of the number *e* raised to the power specified by its argument.

Forms

EXP(power)

Component	*Component Type*	*Component Value*	*Default Condition*
EXP	function name/real	> 0	required component
power	numeric expression	approx -88 to +88	required component

Instructions

Use EXP in any expression requiring an exponential value.

Restrictions

None included.

Errors

?MINC-W-Value of real expression is too small

The power argument is a large negative number and the resulting EXP value is too small to be represented in a real number. MINC assigns the value 0 to EXP and continues.

?MINC-W-Value of real expression is too large

The power argument is too large and the resulting EXP value is too large to be represented in a real number. MINC assigns the value 0 to EXP and continues.

Related

LOG The LOG function takes the value of the natural logarithm of its argument. The following identity expresses the relationship between EXP and LOG.

power = LOG(EXP(power))

Numeric Precision

References

Book 2, Chapter 2.

Examples

Example
```
W=EXP(SQR(2+3))
PRINT W
```

Result 9.35647

Example
```
PRINT EXP(-86)
```

Result 4.47377E-38

EXTRA_SPACE

Purpose

The EXTRA_SPACE statement allows you to increase the space available for your MINC BASIC program. When you execute this statement, MINC adds 2048 words to the workspace.

Forms

EXTRA_SPACE

Component	*Component Type*	*Component Value*	*Default Condition*
EXTRA_SPACE	statement	none	required component

Instructions

EXTRA_SPACE should be used only when READY is displayed. If the program currently in the workspace is to preserved, you should use the SAVE or REPLACE commands before executing EXTRA_SPACE, because a successful operation of this statement causes the workspace to be scratched. Execution of the statement causes some informative text to be displayed, and you are asked if the workspace can be erased. You must respond by typing Y or N. If you enter Y, MINC performs the operation and displays the READY message after a short time. If you enter N, MINC displays a message and then displays the READY message immediately. In this case, the workspace is left intact.

Restrictions

Since EXTRA_SPACE is a statement, it can be executed from a BASIC program. However, MINC never returns to the next program statement. Control will always pass to READY. The penalty you pay for the increased space is that file accessing (OPEN operation) will take slightly longer.

Errors

?MINC-F-Workspace contents remain unchanged

This message is not truly an error message. This message appears when you answer N (no) to the query "Are you ready to have the workspace erased?" See the example below.

Related

NORMAL_SPACE returns the 2048 extra words to BASIC for its use in quick file access.

References

None.

Examples

The following example shows using the EXTRA_SPACE statement.

```
READY

EXTRA_SPACE

Changing the size of the workspace requires erasing
the workspace. You must already have used either the
SAVE or REPLACE command to store the program if
you want to preserve it. Are you ready to have the workspace erased? (Y or N):
Y

READY
```

File Allocation

Purpose

This section discusses how MINC allocates space for a file on a diskette.

Forms

Not applicable.

Instructions

When you close a data file with the CLOSE statement, save a program with a SAVE or REPLACE command, or end an editing session by pressing the STORE FILE key, you are commanding MINC to store a file on a diskette. MINC stores the file on the diskette in the first available area that is large enough to hold the file or in one-half of the largest available area.

If there is not an area large enough to hold the file, MINC does nothing to the diskette and terminates the process with an error message.

Restrictions

If you have a sequential file that is 20 blocks long and there are 30 available blocks on the diskette, you cannot store the file because the largest area available for storage is 15 blocks — one-half of the largest area.

Note that MINC will store a 20-block virtual array file in this case. The one-half restriction applies to sequential files only.

Errors

Not applicable.

Related

OPEN You can use the FILESIZE option of the OPEN statement to command MINC to create a sequential file that is larger than one-half of the available space.

References

None.

Examples

None.

File Specifications

Purpose

The argument filespec appears for many commands and statements in MINC. This section explains the components of a file specification, their meanings, and possible values.

Forms

device:filename.filetype

usually abbreviated as dev:name.typ.

Component	*Component Type*	*Component Value*	*Default Condition*
device	3 characters and :	SY0:,SY1:,LP:	usually SY0: (depends on command)
filename	6 characters	any name	usually required (depends on command)
filetype	. and 3 characters	any type	usually .BAS (depends on command)

The following terms, symbols and abbreviations appear in this summary of MINC's default file specifications.

TERM	*MEANING*
Current name	MINC uses your current program's name unless you specify a different name.
Device	An abbreviation for "device abbreviation"; in each case, MINC processes the volume you have installed in the indicated diskette drive, or the line printer.
Explicit	If you provide an explicit device abbreviation, file name or file type, MINC uses the defaults that appear on the rest of the line.
Input file	The existing file that MINC processes.
File name	The middle part of a complete file specification, a string at least one character long and no longer than six characters, each character being a letter or a digit.
NONAME	MINC processes the file named NONAME unless you specify a different file name.

—	MINC does not accept or use the indicated parts of a file specification.
Output file	The new file that MINC produces.
Required	MINC uses no default for the indicated part of the file specification; you must provide the indicated device abbreviation, file name, or file type.
SY0:	The device abbreviation for system device 0.
Type	The last part of a complete file specification, a string that begins with a period (.), and has from zero to three characters. Note that the null file type is valid. For example, REPORT. is a file named REPORT with a null file type.

The following table shows the default file specifications for all of the BASIC commands and statements.

	INPUT FILE			
COMMAND/ STATEMENT	*DEVICE*	*NAME*	*TYPE*	
APPEND	SY0:	NONAME	.BAS	
DUPLICATE	SY0:	—	—	
CHAIN	SY0:	NONAME	.BAS	
COMPILE	—	—	—	
COMPILE	—	—	—	
COLLECT	SY0:	—	—	
COPY	SY0:	Required	.BAS	
COPY	SY0:	Required	.BAS	
CREATE	SY0:	Required	.BAS	
DIRECTORY	SY0:	All	All	
EDIT	SY0:	Required	.BAS	
INITIALIZE	—	—	—	
KILL	—	—	—	
NAME	SY0:	Required	.DAT	
OLD	SY0:	NONAME	.BAC	(Note 1)
OPEN	SY0:	Required	.DAT	
OVERLAY	SY0:	NONAME	.BAS	
REPLACE	—	—	—	
REPLACE	—	—	—	
RUN / RUNNH	(Note 2)			
SAVE	—	—	—	
SAVE	—	—	—	
TYPE	SY0:	Required	.BAS	
UNSAVE	SY0:	Required	.BAS	
VERIFY	SY0:	—	—	

	OUTPUT FILE			
COMMAND/ STATEMENT	*DEVICE*	*NAME*	*TYPE*	
APPEND	—	—	—	
DUPLICATE	Required	—	—	
CHAIN	—	—	—	
COMPILE	SY0:	Current	.BAC	
COMPILE	SY0:	Explicit	.BAC	
COLLECT	Input's	—	—	
COPY	SY0:	Input's	Input's	
COPY	SY0:	Explicit	.BAS	(Note 3)
CREATE	—	—	—	
DIRECTORY	—	—	—	(Note 4)
EDIT	SY0:	Input's	Input's	(Note 5)
INITIALIZE	SY0:	—	—	
KILL	SY0:	Required	.DAT	
NAME	SY0:	Input's	.DAT	
OLD	—	—	—	
OPEN	SY0:	Required	.DAT	
OVERLAY	—	—	—	
REPLACE	SY0:	Current	.BAS	
REPLACE	SY0:	Explicit	.BAS	
RUN / RUNNH				
SAVE	SY0:	Current	.BAS	
SAVE	SY0:	Explicit	.BAS	
TYPE	—	—	—	
UNSAVE	—	—	—	
VERIFY	—	—	—	

Note 1: MINC searches for the file type .BAC first; if there is none, MINC then searches for the file type .BAS.

Note 2: When you RUN a stored program directly, MINC uses SY0: as the default input device, the name of the file is required and MINC uses the default input file type .BAS .

Note 3: When you specify the output file name without an output file type, MINC uses .BAS for the output file type. When you omit both the output file name and output file type, MINC gives the input file name and type to the output file.

Note 4: Normally, MINC lists a volume directory on your terminal. You can also list the directory on a printer or create an ASCII file of a directory listing. When you print the listing, the output file name and type are not applicable. When you create an ASCII file of a directory, an output file name is required and MINC uses the output file type .DIR unless you specify a different file type.

Note 5: When you omit the output file name and file type in an EDIT command, MINC renames your original input file with

the .BAK file type and gives the original name and type to your edited copy of the file. If you provide either an explicit output file name or an explicit file type, you must provide both explicitly.

The default file types for BASIC are as follows.

.BAS BASIC program.

.DAT Data file — either a sequential or virtual array file.

.BAC A compiled BASIC program.

.BAK A backup file produced by the EDIT command.

Instructions

For each command and statement, read an entire line for full information. Several commands have more than one set of defaults. For example, the default output file type MINC uses in the COPY command depends on whether you include an explicit output file name or not. The two possibilities for the COPY command are shown on separate lines.

Restrictions

See the individual sections for each command or statement.

Errors

See the individual sections for each command or statement.

Related

Protected file types.

References

Book 2.

Examples

See the individual sections for each command or statement.

FOR

Purpose

The FOR statement and the NEXT statement enclose a set of other statements in your program. The set of statements beginning with FOR and ending with NEXT is called a *loop*, and the FOR statement defines the conditions that must exist before statements within the loop are executed.

Forms

FOR control-variable=start-value TO end-value STEP increment

Component	*Component Type*	*Component Value*	*Default Condition*
FOR	statement	none	required component
control-variable	numeric variable name	any whole number	required component
=	none	none	required component
start-value	numeric expression	any whole number	required component
TO	statement component	none	required component
end-value	numeric expression	any whole number	required component
STEP	statement component	none	paired with increment
increment	numeric expression	any whole number	1

Instructions

Each FOR statement defines four features for its loop:

- a counter, called the *control variable;*
- a starting value for the control variable;
- an end value for the control variable;
- a positive or negative step value that MINC uses in counting from the starting value to the end value.

The FOR and NEXT statements determine the boundaries of the loop. The start-value and end-value arguments define the control variable bounds. Before the loop executes once, MINC tests to see if the control variable is within the bounds of the loop. If the control variable is not within the bounds of the loop, control passes immediately to the statement after the NEXT statement. If the control variable is within the bounds, then the loop is executed.

Every time program control reaches the NEXT statement, the control variable is incremented by the step value and then tested to see if it is greater than the end value. If the control variable is

greater than the end value, the step value is subtracted and control passes to the statement after the NEXT statement.

Restrictions

Although MINC accepts numeric expressions for the start-value, end-value, or step arguments, it is better programming practice to use numeric literals or variables. Each time MINC executes a FOR statement, it calculates the values of expressions in the FOR statement first. Therefore, your programs might run noticeably faster if you calculate the required values before the FOR statement.

Be careful when you use a real variable with large values for the control variable that the step specified actually has an effect when it is added to the control variable. Because of limited precision, 1.00000 E+08 does not change when 1 is added to it.

Errors

?MINC-F-No NEXT statement terminates FOR loop at line XX

You forgot to put in a NEXT statement.

Nested loops are improperly nested. See examples.

?MINC-F-Outer FOR loop is using control variable at line XX

More than one loop in nested loops are trying to use the same control variable.

?MINC-F-No corresponding FOR statement for NEXT at line XX

Related

NEXT
Nesting
Loops
Branching
Numeric Precision

References

Book 2, Chapter 6.

Examples

In the following loop, MINC subtracts the step value from J before it exits the loop when J is incremented past 8 (the end-value).

```
10 FOR J=1 TO 8 STEP 2
20 PRINT J
30 NEXT J
40 REM — Print the value of J outside the loop
50 PRINT 'The value of J after the loop is';J
60 END
RUNNH
```

```
 1
 3
 5
 7
The value of J after the loop is 7

READY
```

The following example shows a pair of improperly nested loops and what happens when you try to execute them.

```
READY
LISTNH
10 FOR I=1 TO 10
20 FOR J=1 TO 5
30 NEXT I
40 NEXT J

READY
RUNNH

?MINC-F-No corresponding FOR statement for NEXT at line 30

READY
```

GOSUB

Purpose

Use a GOSUB statement to execute the statements in a subroutine you have written. When MINC executes the GOSUB statement, it transfers control to a subroutine. When MINC executes a RETURN statement within the subroutine, it transfers control back to the statement following the GOSUB instruction.

Forms

GOSUB stmt#

Component	*Component Type*	*Component Value*	*Default Condition*
GOSUB	statement	none	required component
stmt#	numeric literal	1 to 32,767	required component

Instructions

Complete the GOSUB statement with a valid statement number.

MINC accepts GOSUB statements that are within subroutines. Therefore, you can nest subroutines by transferring control from one subroutine into another. MINC's limit is 20 levels of nesting. If 20 subroutines are already active when MINC executes another GOSUB statement, MINC signals an error.

MINC signals an error if the statement number you specify does not exist in your program. GOSUB statements can transfer control to the first statement in a multistatement line but not to subsequent ones.

You cannot use a variable or operator expression as the statement number. MINC does not permit any default for the statement number. MINC does update all GOSUB target statement numbers whenever you use the RESEQ command to renumber statements in a program.

Restrictions

Do not let the main program execute subroutine statements as if they were part of the main program. If this happens, the program halts with an error when it encounters the RETURN statement for the subroutine. See Chapter 9 of Book 2.

Errors

?MINC-F-Reached RETURN without executing a GOSUB statement at line XX

?MINC-F-GOSUB fails; 20 subroutines already active at line XX

You have tried to nest subroutines more than 20 levels deep.

You used a GOTO from a subroutine rather than a RETURN, and the subroutine got called more than 20 times.

?MINC-F-Program does not have a statement number specified at line XX

The statement number specified by the GOSUB statement does not exist in the program.

Related

Branching
ON
IF
RETURN

References

Book 2, Chapter 9.

Book 6, Service Subroutines.

Examples

See Book 2.

GO TO

Purpose

The GO TO statement is one of MINC's branching statements. When MINC executes a GO TO statement, it transfers control immediately to another statement in your program. Therefore, GO TO is the simplest way to transfer control forward or backward in your program.

Forms

GO TO stmt#

Component	*Component Type*	*Component Value*	*Default Condition*
GO TO	statement	none	required component
stmt#	numeric literal	1 to 32,767	required component

Instructions

Complete the GO TO statement with the line number of the statement you want MINC to execute next.

MINC signals an error if the statement number you specify does not exist in your program. GO TO statements can transfer control to the first statement in a multistatement line but not to subsequent ones.

You cannot use a variable or operator expression as the statement number. MINC does not permit any default for the statement number. MINC does update all GO TO target statement numbers whenever you use the RESEQ command to renumber statements in a program.

Restrictions

GO TO statements are simple and attractive, but if you use them heavily in a complicated program, they can make your program difficult to understand and modify. Subroutines, FOR-NEXT loops, CHAIN techniques, and OVERLAY techniques are all easier to follow, if you use them properly.

However, the immediate mode GO TO is extremely convenient while you are debugging a program. For example, after you have corrected a program statement (by retyping it or by using a SUB command), you can use an immediate mode GO TO statement to make MINC continue running your program from the corrected statement or any other statement you choose.

MINC does transfer control to a statement number within a subroutine, but subsequently produces an error when it executes the subroutine's RETURN statement.

Errors

?MINC-F-Program does not have a statement number specified at line XX

The statement number specified by the GO TO statement does not exist in the program.

Related

Branching
IF statements
ON statements

References

Book 2, Chapter 5.

Examples

See Book 2.

HELP

Purpose

The HELP command displays summary information about the system feature you specify.

Forms

HELP subject

Component	*Component Type*	*Component Value*	*Default Condition*
HELP subject	command characters	none see list of subjects when you use default condition	required component list of all available subjects

Instructions

Use HELP when you want a short display of information about some aspect of MINC.

The information is stored in the HELP.TXT file. HELP.TXT is a text file and can be edited using the keypad editor. You can create your own HELP.TXT file following the structural format of the HELP.TXT file supplied with MINC or you can edit the supplied HELP.TXT file to reflect your needs and experience.

The HELP command expects to find HELP.TXT on the volume in SY0:. It ignores a HELP.TXT file on the volume in SY1:.

Restrictions

As the HELP.TXT file grows, the HELP command requires more time to locate information in the file. For speedy assistance, keep edits to the HELP.TXT file concise.

Errors

?HELP-F-Help not available for XXXXXXXXXXX

The HELP command cannot find any information in HELP.TXT using the model you supplied.

?HELP-F-File not found HELP.TXT

The system volume in SY0: is missing the file needed for HELP. The file needed is called HELP.TXT.

Related

Book 8: MINC System Index.

References

Book 1, Chapter 4.

Examples

Example	HELP BASIC
Result	A list of all the BASIC commands and statements appears on the screen.

IF statements

Purpose

IF statements provide the mechanism for conditional branching in a program. Branching is nonsequential transfer of control (see Branching). When the condition you describe in an IF statement is true, MINC executes the branching instruction you have specified.

An IF statement is also a convenient way to execute a nonbranching statement under limited conditions.

Forms

IF true-condition THEN statement

Component	*Component Type*	*Component Value*	*Default Condition*
IF	statement	none	required component
true-condition	logical expression	true or false	required component
THEN	statement component	none	required component
statement	program statement	any valid statement	required component

IF true-condition THEN stmt#
GO TO
GOSUB

Component	*Component Type*	*Component Value*	*Default Condition*
IF	statement	none	required component
true-condition	logical expression	true or false	required component
transfer	statement component	THEN, GOTO, GOSUB	required component
stmt#	numeric literal	1 to 32,767	required component

IF END #channel THEN statement

Component	*Component Type*	*Component Value*	*Default Condition*
IF END	statement	none	required component
#	character	#	required component
channel	numeric expression	1 to 12	required component
THEN	statement component	none	required component
statement	program statement	any valid statement	required component

Instructions

The general logic of an IF statement is as follows:

If a relational expression is true, then execute a statement; otherwise ignore that statement and execute the statement that follows it.

You can specify any valid MINC statement in an IF statement. The following operators are valid in a relational expression.

=	are two values equal?
<>	are two values not equal?
>	is the first value greater than the second?
<	is the first value less than the second?
>=	is the first value greater than or equal to the second?
<=	is the first value less than or equal to the second?

A special and very useful form of the IF statement tests for the end of sequential input files. For example, the following statement executes the subroutine beginning at statement 1000 if there are no more records in the input file associated with channel #3.

```
IF END #3 THEN GOSUB 1000
```

You should be very careful when testing for equality between two real numbers that are not whole numbers. Because of rounding, two values which may be close enough for your purposes might be unequal to the IF statement.

Restrictions

If you use the IF statement as the first statement on a multiple-statement line, the results can be very confusing. The IF statement in conjunction with the backslash follows the rules listed below.

IF logical-expression THEN stmt1 \ stmt2 \ stmt3

With this form of the IF statement, the statements following the backslash are executed if the logical expression is true. That is, if the logical expression is true, then MINC executes statements 1, 2, and 3. If the logical expression is false, then MINC does not execute any of the three statements but rather proceeds to the statement following the IF statement.

IF logical-expression GO TO stmt# \ stmt1 \ stmt2
IF logical-expression THEN stmt# \ stmt1 \ stmt2

With this form of the IF statement, the statements following the backslash are executed if the logical expression is false. That is,

- If the logical expression is true, then MINC jumps to the statement number specified in the GO TO phrase, and statements 1 and 2 are not executed.
- If the logical expression is false, then MINC does not jump to the statement number specified in the GO TO phrase; instead it executes statements 1 and 2 and then proceeds to the statement following the IF statement.

Errors

There are no error messages associated with IF statements.

Related

Branching
ON statements
Program structures

References

Book 2, Chapter 5.

Examples

See Book 2.

INITIALIZE

Purpose

Use the INITIALIZE command to prepare a new volume for use on MINC and to erase all files on a used volume. You must initialize a volume before using it with the DUPLICATE command.

Forms

INITIALIZE SY1:

Component	*Component Type*	*Component Value*	*Default Condition*
INITIALIZE	command	none	required component
SY1:	characters	none	required component

Instructions

When you complete the command, MINC reports the current volume identifier and owner. You cannot initialize a volume whose "owner" is DIGITAL. MINC prompts:

```
Proceed with initialization (Y or N)?
```

Type NO to cancel the command; type YES to confirm that MINC should destroy all files on the volume you have loaded and proceed with the initialization.

MINC's first step in processing the command is to ask for a new volume identifier and owner name. The identifier and name can use any characters on your main keyboard except control combinations, and they can be from 0 to 12 characters long.

When you have entered the new volume identifier and owner name, MINC erases the entire directory of the volume you are initializing. MINC then checks each block on the volume, marks each bad block so that you cannot use the block accidentally and counts the number of bad blocks. If MINC finds more than 20 bad blocks, it displays a message that the volume is unusable. For 20 bad blocks, or less, MINC displays the following message:

```
Volume initialized. nnn bad blocks found.
```

Restrictions

Choose volume identifiers that are informative as well as unique. For example, a volume identifier that indicates the general purpose of most files to be stored on the volume helps to distinguish it from other volumes you are using. For a system diskette, choose a volume identifier which reflects the fact that the

volume is a system diskette because system files do not appear in a directory listing.

Assign the owner equally carefully. If you are using several different volumes in your work, you might find it useful to assign different owner names depending on the purpose for the volume.

Errors

?UTILITY-F-Unable to initialize volume with owner name 'DIGITAL'

You mistakenly put a Master diskette in SY1: instead of a new diskette.

?UTILITY-F-Error writing directory

The diskette in SY1: has bad blocks in the directory. Use another diskette.

?UTILITY-F-Read error

There is no diskette in SY1:, or the diskette is inserted incorrectly.

?UTILITY-F-Device SY may not be initialized.

You cannot initialize SY0:. The only form of the command is INI SY1:.

Related

DUPLICATE
VERIFY

References

Book 2, Chapter 4.

Examples

The following example shows the procedure for initializing a volume.

```
READY
INI SY1:
Install volume to be initialized in SY1, and press RETURN (RET)

Current volume id:eeeeeeeeeeee
Current owner: eeeeeeeeeeee
Proceed with initialization (Y or N)?Y
Type new Volume id Lab 3 system
Type new owner name:Jane Doe

Initialization is complete; found 000 bad blocks
```

Now you can use this new volume to store programs or a system or data.

INPUT

Purpose

Use an INPUT statement whenever you want your program to get another data value or set of values from a sequential data file or from your keyboard. When the INPUT statement specifies a channel number, MINC reads the next values from the file associated with the channel. When the INPUT statement does not specify a channel number, MINC displays a question mark (?) on your screen and processes the values you type in response.

Forms

INPUT # channel, variable-list

Component	*Component Type*	*Component Value*	*Default Condition*
INPUT	statement	none	required component
#	character	#	paired with channel
channel	numeric expression	0 to 12	input from keyboard
variable-list	real, integer, or string	variable name	required component

The variable list can include array elements as well as real, integer, and string variables. The channel number must have a numeric value in the range 0 through 12. When the channel number is equal to zero, MINC takes values that you type on your keyboard, but it does not display the question mark (?) prompt for input, in this case. When the channel number is omitted, MINC displays the question mark prompt and takes values that you type on your keyboard.

Instructions

Complete the INPUT statement with a list of variables separated by commas. When MINC executes the statement, it assigns the data values to the variables in the same order. Therefore, real, integer, and string values must be in the proper order to match corresponding real, integer, and string variables or array elements.

MINC accepts data values for INPUT statements when there is one value on each data line and when there are several values on a data line. When you have one value on each data line, MINC processes one line for each INPUT statement variable or array element. When you have several values on a single line, MINC processes the line from left to right, collecting the characters for each value until a comma is encountered. When you have too many values on a data line for the number of variables and array elements in an INPUT statement, MINC ignores the extra values and displays a warning.

You need not place string input values in matching quotes. MINC accepts matching quotes and you need them if the string contains a comma. When you mix string values and numeric values on a data line, check that a comma follows the closing quote for each string.

When you want to use an INPUT statement to receive values from the keyboard, include an explanatory PRINT statement immediately before the INPUT statement. The PRINT statement can describe how many values to type and their order.

For inputting string information, use the LINPUT statement.

Restrictions

An INPUT statement uses a complete line of input from either a file or the terminal. Thus, if there are more values on the line than variables in the INPUT statement, the subsequent values are lost.

Errors

?MINC-W-Extra values from keyboard or file ignored at line XX

You entered more values on a line (separated by commas) than INPUT variables.

There were more values on a line of a sequential file than INPUT variables.

?MINC-W-Enter new value. Old value did not match INPUT variable at line XX

The input value is a string and the INPUT variable is numeric, or the input value is a number and the INPUT variable is a string variable. Note that MINC then waits for you to enter an appropriate value; however, it does not prompt you to enter a value.

?MINC-F-Too few values for INPUT or READ variables at line XX

There were not enough values in a sequential file for the number of INPUT variables.

?MINC-F-Need OPEN statement for file channel at line XX

You put an INPUT # statement in your program without opening the channel associated with the file.

Related

LINPUT
RESET
RESTORE
PRINT

READ The READ statement requires that the corresponding data values be in your workspace with your current program. READ statements take data values only from DATA statements. INPUT statements take only data values that are not in your workspace.

DATA
OPEN
CLOSE
IF END #

References

Book 2, Chapters 3 and 11.

Examples

See Book 2.

INSPECT

Purpose

The INSPECT command enables you to inspect a file of characters using the keypad editor. You can use any of the keys which move the cursor but none of the keys which would change the contents of the file.

Forms

INSPECT filespec

Component	*Component Type*	*Component Value*	*Default Condition*
INSPECT	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name required .typ .BAS

Instructions

Use the INSPECT command when you want to determine the contents of a file without changing them in any way.

Press the STORE FILE key when you have finished inspecting the file. Because the file is unchanged, MINC does not store a new version of the file or create a backup file (type .BAK).

Restrictions

You should not inspect virtual array files, .BAC files, or any other non-ASCII files.

Errors

If you press any keys other than the cursor movement keys or CTRL/C, the terminal sounds its warning tone to remind you that you cannot change the file.

Related

CREATE The CREATE command invokes the editor. You can create a new file by typing it on the keyboard.

EDIT The EDIT command invokes the keypad editor. You can modify an existing file or create a new file which is a modified version of an existing file.

LIST The LIST command lists the contents of the workspace. You control which program statements appear with the statement number arguments to LIST.

TYPE The TYPE command displays a file on the screen starting at the beginning of the file and continuing to the end. The NO SCROLL key suspends and restarts this display. The editor cursor commands have no effect.

References

Book 2, Chapter 15.

Examples

See Book 2.

INT

Purpose

The INT function takes the value of the largest whole number (integer) less than or equal to the value of its argument.

Forms

INT(number)

Component	*Component Type*	*Component Value*	*Default Condition*
INT	function name/numeric	any whole number	required component
number	numeric expression	any value	required component

Instructions

For a positive argument, INT truncates any fraction and takes the value of the whole number part of the argument. For a negative argument, INT again takes the whole number value *smaller* than the argument.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

None.

References

Book 2, Chapter 8.
Book 2, Mixed Mode Arithmetic.

Examples

```
Example   PRINT INT(10.3)
Result      10

Example   A=INT(-7.2)
          PRINT A
Result    -8

Example   PRINT INT(-23)
Result    -23
```

Keypad Editor

Purpose

The MINC keypad editor allows you to edit programs and ASCII files quickly and easily. Although you can edit BASIC programs by using BASIC commands, there is no easy way to edit a file that contains text, such as a letter, memo, or mailing list, easily with BASIC. The keypad editor enables you to create, inspect, and edit text files. You can also use the editor to fix BASIC programs that were damaged when your volume developed bad blocks.

Forms

CREATE filespec
EDIT inputfilespec outputfilespec
INSPECT filespec

Instructions

Refer to the sections for the individual commands.

Use the CREATE command to create a new file by entering text at the keyboard.

Use the EDIT command to modify an existing file or to create a new file which is a modified version of an existing file.

Use the INSPECT command to display an existing file on the screen without permitting modifications.

Restrictions

You can use the keypad editor to edit BASIC programs. However, the editor accepts any text as input and you can modify a program so that BASIC cannot translate it. BASIC cannot load or run a program with missing statement numbers or with spurious or invisible characters. When you try to run an invalid program, BASIC stops loading the program into the workspace when it detects a statement it cannot recognize and then reports a syntax error. Thus, if you have an error in the first line of the program, BASIC does not load any of the program into the workspace. In this case, you must use the editor to determine the problem and fix it.

Do not edit virtual array files or any non-ASCII files such as .BAC files.

Some of the control characters have different meanings to the editor than they do to MINC. These control characters are as follows.

CTRL/O The CTRL/O character stops all output and input to the terminal. A second CTRL/O character resets the terminal.

Note that when you press the CTRL/O character the first time, you might get a spurious character on the screen. However, this character goes away when you reset CTRL/O.

When you reset CTRL/O, the cursor is not always in the same place on the screen as MINC thinks it is internally. CTRL/W refreshes the screen so that the cursor is in the same place on the screen as it is internally.

CTRL/S, CTRL/Q These characters work the same in the editor as in BASIC.

CTRL/U The CTRL/U character deletes from the current cursor position to the beginning of the line. Note that this operation is different than the CTRL/U operation in BASIC.

Errors

See the individual commands for the errors that apply to them.

Related

SUB
DEL

References

Book 2, Chapter 15.

Examples

See Book 2.

KILL

Purpose

Use the KILL statement within a program to erase a file on a volume you are currently using. The KILL statement is the program statement that is equivalent to the UNSAVE command. Although KILL erases any valid file, the best uses for KILL statements are to erase large data files that are no longer important and temporary files you have finished using.

After MINC executes a KILL statement, there is no way for you to recover the file you have erased.

Forms

KILL "filespec"

Component	*Component Type*	*Component Value*	*Default Condition*
KILL	statement	none	required component
filespec	string expression	dev:name.typ	dev: SY0: name required .typ .DAT

Instructions

Complete the KILL statement with the name of the file you want to erase. Enclose the file name in matching quotation marks.

Restrictions

Note that the KILL statement defaults to SY0: and the .DAT extension, whereas the UNSAVE command defaults to SY0; and the .BAS extension. However, you can delete any kind of file with either command.

Errors

?MINC-F-specified or default volume does not have file named

Related

UNSAVE
NAME
File specifications

References

Book 2, Chapter 11.

Examples

See Book 2.

LEN

Purpose

The LEN function takes on a value corresponding to the number of characters in its string argument.

Forms

LEN(string)

Component	*Component Type*	*Component Value*	*Default Condition*
LEN	function name/numeric	0 to 255	required component
string	string expression	any valid string	required component

Instructions

Use LEN to determine how many characters are in a string.

LEN is useful for string processing, for example, screening input to check that the required number of characters is present or centering string displays on the screen.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called

You did not use a string expression as the argument.

Related

**POS, SEG$** The POS and SEG$ functions perform related string processing operations. The POS function determines the location of a search string within another string. The SEG$ function extracts a substring from within another string.

References

Book 2, Chapter 8.

Examples

```
Example  PRINT LEN('abcdefghijklmnopqrstuvwxyz')
Result   26

Example  A=LEN('MINC')
         PRINT A
Result   4

Example  PRINT LEN('')
Result   0
```

LENGTH

Purpose

Normally, MINC provides approximately 5000 words in the workspace. A large program or a series of program files combined by APPEND commands can exhaust your workspace and cause MINC to display messages such as

```
?MINC-F-Array overfills workspace at line 10
```

The LENGTH command reports how many words are used and how many words are still available in the workspace. The LENGTH command subtracts the length of the current program and the arrays and variables allocated for it from the full size of the workspace.

Forms

LENGTH

Component	*Component Type*	*Component Value*	*Default Condition*
LENGTH	command	none	required component

Instructions

Whenever MINC is READY, use the LENGTH command to check the workspace in use by your current program and to get a report of the amount of workspace that is not being used. Type LENGTH whenever MINC displays READY.

Restrictions

To determine the total size of the workspace, enter the LENGTH command after performing an SCR command.

Errors

There are no error messages associated with this command.

Related

EXTRA_SPACE
NORMAL_SPACE

References

Book 2, Chapter 4.

Examples

See Book 2.

LET

Purpose

MINC accepts the verb LET to distinguish assignment statements from other statements, but there is no important benefit from using LET. For example, the two statements LET A%=5 and A%=5 mean and do the same thing: "store the value 5 in the integer variable named A%." Some MINC users prefer to use LET in every statement that assigns a value to a variable or array element, but most MINC users choose to save workspace by leaving LET out.

Forms

LET variable = value

Component	*Component Type*	*Component Value*	*Default Condition*
LET	statement	none	assigns value
variable	string or numeric variable name	any valid name	required component
=	none	none	required component
value	string or numeric expression	same type as variable	required component

Instructions

Use LET in any assignment statement — any statement that assigns a value to a variable or array element — if you like.

Restrictions

None.

Errors

There are no error messages associated with this statement.

Related

Assignment statements

References

Book 2, Chapters 2 and 3.

Examples

See Book 2.

LINPUT

Purpose

Use a LINPUT statement to get an entire data line from a sequential data file or from your keyboard. LINPUT treats each entire line as a character string. MINC takes a line of data for each variable or array element in the LINPUT statement.

Techniques for using LINPUT statements are often more complex than equivalent techniques for INPUT statements. LINPUT statements are somewhat more convenient than INPUT statements for string values that are on separate lines. The principal advantages in this case are that you do not have to enclose a string value in string delimiters and you do not need special techniques to process the apostrophe and quotation mark characters. However, to input numeric values, you have to use special conversion functions, such as VAL, to evaluate the numeric equivalents of the strings of digits LINPUT takes from a file.

Forms

LINPUT # channel, variable-list

Component	*Component Type*	*Component Value*	*Default Condition*
LINPUT	statement	none	required component
#	statement component	#	paired with channel
channel	numeric expression	0 to 12	read from keyboard
variable-list	string variable name	any string variable	required component

The value of the channel number must be in the range 0 through 12. When the channel number is equal to zero, MINC receives values from the keyboard.

Instructions

Complete a LINPUT statement with a list of string variables separated by commas. When MINC executes a LINPUT statement, it gets an entire line from a file or from your terminal for each variable or array element in the LINPUT statement. Every character in each line is significant. MINC takes all commas, apostrophes, and quotation marks as literal characters. From the keyboard, the RETURN key terminates the line. When LINPUT encounters the end of the line, it assigns the contents of the line as the value of the current string variable and then either finishes executing or reads the next line if there are more variables to be filled.

When you do not specify a channel number in a LINPUT statement, MINC displays a question mark prompt on your screen.

Each line you type in response is assigned to the corresponding variable or array element in the LINPUT statement.

When you specify channel number zero in a LINPUT statement, MINC also reads the characters entered on the keyboard but, in this case, does not display the prompt.

Restrictions

None included.

Errors

?MINC-F-Too few values for INPUT or READ variables at line XX

There were not enough values in a sequential file for the number of variables in a LINPUT statement.

?MINC-F-Syntax error; cannot translate the statement at line XX

A LINPUT variable is not a string variable.

?MINC-F-Need OPEN statement for file channel at line XX

You used a LINPUT # statement without using an OPEN statement to open the associated file.

Related

INPUT

References

Book 2, Chapter 3.

Examples

See Book 2.

Line Printer

Purpose

The line printer is an optional MINC device that allows you to obtain information from MINC on paper. The line printer has the device name LP:. Several commands and statements permit LP: as a file specification.

Forms

OPEN 'LP:' FOR OUTPUT AS FILE x
PRINT # x ...

DIR filespec LP:

SAVE LP:

COPY filespec LP:

Instructions

Using the forms of the commands listed above directs the output to the line printer instead of to the terminal or a file.

Restrictions

The line printer is only an output device.

Errors

See the individual commands for the appropriate errors.

Related

DIR
COPY
OPEN
SAVE

References

Book 2.

Examples

The following example prints "This is a test of the line printer" on the line printer.

```
10 OPEN 'LP:' FOR OUTPUT AS FILE 1
20 PRINT #1, 'This is a test of the line printer'
30 CLOSE
40 END
```

LIST/LISTNH

Purpose

When MINC displays READY, use the LIST command (or LISTNH) to display all or parts of your current program on your screen. MINC shows each statement in the system's internal form. BASIC verbs and other program keywords appear in capital letters, and all letters in variable and array names are also capitalized. String literals, the text in REM statements, and MINC routine names appear as you entered them, in upper or lower case. Strings of spaces and tabs are collapsed to a single space except within string literals or remarks. MINC always lists your program in statement number order.

Forms

LISTNH linespec, linespec, ...

Component	*Component Type*	*Component Value*	*Default Condition*
LISTNH	command	none	required component
linespec	none	see form of linespec	entire program
Form of linespec:			
firstline	numeric literal	1 to 32,767	first program statement
-	statement component	-	paired with lastline
lastline	numeric literal	1 to 32,767	last program statement

Instructions

MINC always reports program errors with the statement number of the error (or of a closely related statement). You can use the LIST command (or LISTNH) to display the single statement or a small group of statements in order to determine the error.

The only difference between LIST and LISTNH is that LISTNH does not display a heading before the first statement of your program. With slow speed terminals, there was a speed advantage to using LISTNH. However, with the fast MINC terminals, LIST has the advantage of showing identifying material for the program.

If you do not specify statement numbers with LIST (or LISTNH), MINC displays the entire program. (The section entitled "CTRL Operations" describes how to interrupt a long listing.)

If you want to display parts of the current program, add statement numbers to the LIST command (or LISTNH). You can list

a single statement, a series of nonconsecutive statements, a group of consecutive statements, several groups, or any combination.

Note that if you try to list a line or lines that are not in your program, MINC does not give you an error message. MINC prints the header line and then a blank line.

Restrictions

You may find it useful to limit the listings you ask for to about 15 statements. Your screen will then have enough room for you to enter, retype, or correct a few statements before any statements you have displayed scroll off the top of your screen.

You can review the statements in the main program surrounding a GOSUB statement as well as relevant lines from a subroutine at the same time by specifying two statement ranges in a LIST command.

The LIST command is the only way to examine the program currently in your workspace. The TYPE command works for diskette files only.

Errors

There are no error messages associated with this command.

Related

CTRL operations
COPY
SAVE
TYPE

References

Book 2, Chapter 4.

Examples

Example LIST 10

Result Lists line 10 only.

Example LIST 150-350

Result Lists all lines between 150 and 350.

Example LIST 200-

Result Lists all lines from 200 to the end.

Example LIST

Result Lists the entire program.

LOG

Purpose

The LOG function takes the value of the natural logarithm of its argument.

Forms

LOG(number)

Component	*Component Type*	*Component Value*	*Default Condition*
LOG	function name/real	-87 to 88	required component
number	numeric expression	any positive real number	required component

Instructions

Use LOG to determine the natural logarithm of an expression. The LOG function is equivalent to the following mathematical expression:

$$\log_e(\text{number})$$

The base of the natural logarithms, e, has the value approximately 2.71828 in MINC.

Restrictions

None included.

Errors

?MINC-W-LOG or LOG10 expression less than or equal to 0 at line XX

Related

The LOG function is the inverse of the EXP function. The following identity expresses the relationship between LOG and EXP:

$$\text{number} = \text{LOG}(\text{EXP}(\text{number}))$$

The LOG function can be used to generate a logarithm for any required base, as shown in the following identity:

$$\log_x(\text{number}) = \log_e(\text{number})/\log_e(x)$$

References

Book 2, Chapter 2.
Book 3, Numeric Precision.

Examples

Example PRINT LOG(2.718281)

Result 1

Example A=LOG(2.71829)
PRINT A

Result 1

Example PRINT LOG(2.71828)

Result .999999

LOG10

Purpose

The LOG10 function takes on the value of the base-10 logarithm of its argument.

Forms

LOG10(number)

Component	*Component Type*	*Component Value*	*Default Condition*
LOG10	function name/real	-38 to 38	required component
number	numeric expression	any positive real number	required component

Instructions

Use LOG10 to determine the base-10 logarithm of any numeric expression.

Restrictions

None included.

Errors

?MINC-W-LOG or LOG10 expression less than or equal to 0 at line XX

Related

LOG The LOG function provides the natural logarithm of any numeric expression. The LOG10 and LOG functions are related by the following identity:

LOG10(number) = LOG(number)/LOG(10)

References

Book 2, Chapter 2.

Examples

```
Example   PRINT LOG10(10)
Result      1

Example   A=LOG10(1000)
          PRINT A
Result      3

Example   PRINT LOG10(1E-6)
Result    -6
```

Messages

Purpose

MINC is a system with many related components. Although each of the components has been both carefully designed and thoroughly tested, some mistakes and faults are inevitable. A diskette might be damaged but still seem to be in working order — until you try to use it. A connection could be loose or unplugged, or an electrical component could fail. At a different level, you might make a typing mistake or a logical error. You might fill a diskette to capacity and then try to store a file that cannot fit on it.

For each of those conditions (and many more), MINC displays a brief message that describes what has happened. There are hundreds of messages, many of which you might never see and others you are likely to see often.

Book 8: MINC System Index lists and explains all of the messages your MINC system can send you. Read the beginning of Book 8 carefully for full details about how to use it. For your convenience, this section summarizes the most important message features.

Forms

?environment-severity-text

Component	*Component Type*	*Component Value*	*Default Condition*
environment	characters	MINC, EDITOR, DIR, UTILITY, KMON, MON, BOOT	always present
-	character	hyphen	always present
severity	character	F, W, or I	always present
-	character	hyphen	always present
text	characters	explanation	always present

The forms of prompt sequences and reports depend on the commands and processes they pertain to.

Instructions

F stands for fatal message. MINC stops your program when you get a fatal error message.

W stands for warning message. MINC does not halt your program in this case, although you should fix the cause of the warning and run the program again.

I stands for information. These messages give you information about the environment but do not halt the program.

When you see an unfamiliar error message, look it up either in Book 8 or in the book that seems to be relevant. The discussion of the message in Book 8 directs you to the relevant book for more information.

In Book 8, all messages are alphabetized in the same form as they appear on the screen. The order for any special characters appears in the ASCII collating sequence in Appendix A of Book 2.

The most frequent errors occur in program statements. The messages relevant to these errors have the environment designator MINC. Less common errors involve the system utility programs, the directory, and the keypad editor. These messages have environment designators UTILITY, DIR, and EDITOR, respectively.

The least frequent errors occur through some of the MINC system files. These messages have environment designators KMON, MON, and BOOT. The BOOT messages appear only if you try to start MINC with a diskette that is not a system diskette or a diskette with bad system files.

The description for every message in the manual includes suggestions for you to follow when the message appears. Some messages also have highly technical descriptions and instructions that involve MINC's design and the ways MINC works. Those messages are unlikely to appear, however, and if any of them does appear, it probably signals a problem that requires technical service.

After most messages, MINC returns to the READY message, and you can try your procedure again. Occasionally, however, you may have to restart your system.

Whenever you see a message that is not covered in the message manual, your system has developed a serious problem. If this should happen, notify the MINC Product Service Center at the following telephone numbers.

Massachusetts customers:	1-800-762-9700
Customers in rest of continental United States:	1-800-225-9366

Customers in Alaska, Hawaii, and Canada: 1-617-493-9473

Restrictions

None included.

Errors

Not applicable.

Related

None included.

References

Book 2.
Book 8.

Examples

None included.

NAME

Purpose

By using a NAME statement in a program, you can change the name of any file that is not currently open.

Forms

NAME "old-filespec" TO "new-filespec"

Component	*Component Type*	*Component Value*	*Default Condition*
NAME	statement	none	required component
old-filespec	string expression	dev:name.typ	dev: SY0: name required .typ .DAT
TO	statement component	none	required component
new-filespec	string expression	dev:name.typ	dev: SY0: name old-filespec name .typ .DAT

Instructions

Complete the NAME statement with the existing file name and the new name you are giving to the file. If you use an explicit device abbreviation for either file name, you must also use it for the other one.

The NAME statement requires careful labeling and use of diskettes as well as thorough testing of the programs that use it. In particular, be careful to install the correct volume in the proper device before you run a program that has a NAME statement. If the output volume already has another file with the new name you are assigning, that file will be erased. MINC does not give any warning when this happens, and there is no way to recover a file that you accidentally erase with a NAME statement. For example, if your SY1: volume already contains a file named ROSTER.C41, the following statement erases it:

```
NAME "SY1:LIST.TXT" TO "SY1:ROSTER.C41"
```

After MINC executes the statement, the contents of LIST.TXT have the name ROSTER.C41.

Restrictions

If the device does not match in both file specifications, the wrong file can get named. For example,


```
NAME 'PROG.BAS' TO 'SY1:NEW.BAS'
```

This NAME statement renames SY0:PROG.BAS to SY0:NEW.BAS. The volume in SY1: is not affected.

Errors

?MINC-F-Specified or default volume does not have file named

?MINC-F-I/O error; unable to check volume owner

The diskette in SY1: is inserted incorrectly or is missing.

Related

RENAME The RENAME command changes the name of the program in the workspace.

References

Book 2.

Examples

Example	NAME 'PROG.BAS' TO 'PROG.OLD'
Result	The program in file SY0:PROG.BAS has been renamed to SY0:PROG.OLD.

NEW

Purpose

When MINC displays READY, use the NEW command to erase your workspace and provide the name of a new program you want to type in. When MINC executes a NEW command, it deletes all program lines in the workspace and cancels all of the current variables, arrays, and functions you have defined in DEF statements.

Forms

NEW filename

Component	*Component Type*	*Component Value*	*Default Condition*
NEW	command	none	required component
filename	characters	SY0:name.BAS	prompts for name

Instructions

If you want to store your current program, use the SAVE command or the REPLACE command before typing NEW. After MINC executes a NEW command, you cannot recover any program lines that MINC erased.

If you specify a program name with the NEW command, MINC uses the name. If you do not specify a new name, MINC requests one with the following message.

```
NEW FILE NAME—
```

If you respond with a name, MINC uses it. If you respond by pressing only the RETURN key, MINC uses the name NONAME.

The NEW command always assumes device SY0: and file type .BAS, regardless of what you enter for device and file type. Thus, with the NEW command, all you can provide is the name, not the device or the file type.

Restrictions

If the name argument has more than six characters, NEW truncates to six with no warning.

If you entered a file name and a file type, NEW stops reading when it finds the dot (.) and ignores whatever file type you tried to enter. For example, the following sequence of commands name the workspace ABCDEF.BAS.

```
READY
NEW ABCDEFGH.EX1

READY
LIST

ABCDEF 10:20:15 17-JUL-78

READY
```

Errors

?MINC-F-Invalid file name

There are invalid characters in the file name. In this case, MINC defaults the name to NONAME.

Related

SCR
CLEAR
SAVE
REPLACE

References

Book 2, Chapter 3.

Examples

Example NEW SINES

Result The workspace is now named SINES.BAS.

NEXT

Purpose

Every FOR statement you use must have a corresponding NEXT statement that follows it somewhere in the program. The paired FOR statement and NEXT statement define a loop. The FOR statement controls how many times MINC executes the set of statements in the loop. Each time the NEXT statement for a loop executes, the program transfers control back to the FOR statement, increments the loop control variable, tests the new value of the control variable, and repeats the loop's statements if the control variable is still in the proper range of values.

Forms

NEXT control-variable

Component	*Component Type*	*Component Value*	*Default Condition*
NEXT	statement	none	required component
control-variable	numeric variable name	NEXT assigns value	required component

Instructions

Complete the NEXT statement with the control variable that the corresponding FOR statement uses. A NEXT statement cannot precede its FOR statement. MINC does accept NEXT statements that are in multistatement lines.

In certain cases it is useful to use a GO TO statement within a loop to transfer control to the NEXT statement at the end of the loop. With that technique, you can execute some statements in the loop without executing them all.

Restrictions

Be careful when you put a NEXT statement in a THEN clause. The following example loops only once.

```
10 FOR I=1 TO 10000
20 IF I>50 THEN NEXT I
```

Errors

?MINC-F-NO NEXT statement terminates FOR loop at line XX

You forgot to put in a NEXT statement.

Nested loops are improperly nested.

?MINC-F-Outer FOR LOOP is using control variable at line xx

More than one loop in nested loops are trying to use the same control variable.

?MINC-F-No corresponding FOR statement for NEXT at line XX

Related

FOR
Loops
Branching

References

Book 2, Chapter 6.

Examples

See FOR.

NORMAL_SPACE

Purpose

The NORMAL_SPACE statement is the opposite of the EXTRA_SPACE statement. When you execute this statement, MINC removes 2048 words from the workspace to allow faster file access.

Forms

NORMAL_SPACE

Component	*Component Type*	*Component Value*	*Default Condition*
NORMAL_SPACE	statement	none	required component

Instructions

NORMAL_SPACE should be used only when READY is displayed. If the program currently in the workspace is to be preserved, you should use the SAVE or REPLACE commands before executing NORMAL_SPACE, because a successful operation of this statement causes the workspace to be scratched. Execution of the statement causes some informative text to be displayed, and you are asked if the workspace can be erased. You must respond by typing Y or N. If you enter Y, MINC performs the operation and displays the READY message after a short time. If you enter N, MINC displays a message and then displays the READY message immediately. In this case, the workspace is left intact.

Restrictions

Since NORMAL_SPACE is a statement, it can be executed from a BASIC program. However, MINC never returns to the next program statement. Control will always pass to READY.

Errors

?MINC-F-Workspace contents remain unchanged

This message is not truly an error message. This message appears when you answer N (no) to the query "Are you ready to have the workspace erased?" See the example below.

Related

EXTRA_SPACE adds the 2048 extra words to the workspace.

References

None.

Examples

The following example shows the use of the NORMAL_SPACE statement.

```
READY
NORMAL_SPACE

Changing the size of the workspace requires erasing
the workspace. You must already have used either the
SAVE or REPLACE command to store the program if
you want to preserve it. Are you ready to have the workspace erased? (Y or N):
Y

READY
```

Numeric Precision

Purpose

The purpose of this section is to help you understand why MINC does not always display results that are accurate to as many decimal places as you would like.

Forms

Not applicable.

Instructions

MINC calculates numbers to only about 1 in 10^7 parts. For this reason, numbers that cannot be represented exactly with this precision can only be imprecisely represented.

No irrational number (such as π) can be exactly represented numerically by any computer, including MINC.

Note that sin(x) is approximately equal to x for all x's very close to 0, π, 2π, and so forth. Therefore, if π is incorrect by 1 in 10^7, then $\sin(\pi)$ is approximately equal to $1/10^7$.

Similar reasoning applies to many other problems. For example, in a computer,

$$(\sqrt{5})^2 \neq 5$$

because $\sqrt{5}$ cannot be represented exactly. By the same token,

$$3*(1/6) \neq .5$$

because 1/6 cannot be represented exactly.

Note that the rounding process cannot help, because when a number cannot be represented accurately, neither you nor the computer can guess how to round it.

Restrictions

None included.

Errors

None included.

Related

None included.

References

Knuth, D. E., *The Art of Computer Programming.* Volume 2, section 4.2. Reading, Mass.: Addison-Wesley, 1969.

Examples

None included.

OCT

Purpose

The OCT function takes on a numeric value equivalent to an octal value specified as a string of digits whose values range from 0 to 7. The OCT function provides a method for converting a string representing an octal value to its numeric equivalent. There is no reverse operation for converting a numeric value to an octal string.

Forms

OCT(octal-string)

Component	*Component Type*	*Component Value*	*Default Condition*
OCT	function name/integer	-32,768 to 32,767	required component
octal-string	string expression	'0' to '177 777'	required component

Instructions

Use OCT when you need to manipulate the numeric equivalent of an octal value.

Restrictions

The only characters permitted in the octal-string argument are 0 through 7 and space.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

If converted, the octal-string argument would result in a value for OCT exceeding the integer range.

Invalid characters are in the string.

Related

BIN performs a similar conversion for a string of 1's and 0's representing a binary value.

References

None.

Examples

Example PRINT OCT('177 777')

Result -1

Example
```
A=OCT('1000')
PRINT A
```

Result 512

Example PRINT OCT('77 777')

Result 32,767

Example PRINT OCT('100 000')

Result -32,768

Example PRINT OCT('')

Result 0

OLD

Purpose

Use the OLD command to bring a program from a volume you are using into your workspace. When MINC executes an OLD command, it erases your workspace entirely, clears all variables, arrays, and functions you have defined, and then gets the program from the file you have specified.

Forms

OLD filespec

Component	*Component Type*	*Component Value*	*Default Condition*
OLD	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name prompts for name .typ .BAC first, then .BAS

OLD searches for the file type .BAC first. If the input volume does not have a file with the specified name and type .BAC, OLD searches for the file type .BAS.

Instructions

If you want to store your current program, use the SAVE command or REPLACE command before you type OLD. After MINC executes an OLD command, you cannot restore any statements that MINC erases.

If you specify a file name in an OLD command, MINC brings the program in the file into your workspace directly. If you do not specify a file name, MINC requests a file name with the following message.

OLD FILE NAME—

MINC brings the program you specify into your workspace.

Restrictions

None included.

Errors

?MINC-F-Specified or default volume does not have file named

The file is not on the volume.

The volume has bad blocks in the directory.

There is no volume in the drive.

?MON-F-Trap to X XXXXX

The specified volume is uninitialized.

?MINC-F-Invalid file name

There are invalid characters in the file named.

?MINC-F-Syntax error; cannot translate the statement

Editing with keypad editor left a blank line in the file or a line number with a blank statement. Both are fatal to OLD.

Related

APPEND
CHAIN
Keypad Editor
OVERLAY
SAVE
REPLACE

References

Book 2, Chapter 4.

Examples

See Book 2.

ON statements

Purpose

Use an ON statement in a program when you want the result of an expression to define and control a multiple branch. The GO TO statement and GOSUB statement by themselves permit only an unconditional branch to a single statement number. ON statements combine with GO TO and GOSUB statements to provide unconditional branching to one of a set of statement numbers. The particular statement number branched to is selected by the value of a control expression.

Forms

ON control-value GO TO stmt#-list
GOSUB

Component	*Component Type*	*Component Value*	*Default Condition*
ON	statement	none	required component
control-value	numeric variable or expression	1 to list length	required component
GO TO GOSUB	statement component	none	required component
stmt#-list	list	1 to 32,767	required component

Instructions

Use the ON statement to control multiple conditional branching based on a numeric value.

Separate the target statement numbers with commas. The order of the statement numbers controls the branching behavior of the program.

If the control value is negative, equal to 0, or has a larger value than the number of target statements, MINC prints the following message.

```
?MINC-F-Value of control expression is out of range at line 10
```

Therefore, you should test the control value immediately before any ON statement and include other statements that handle improper control values.

When you call a subroutine with an ON GOSUB statement, the RETURN statement always returns control to the statement following the ON GOSUB statement.

Restrictions

None.

Errors

?MINC-F-Value of control expression is out of range at line XX

The value of the control expression is less than 1 or greater than the number of statement numbers in the list.

?MINC-F-Program does not have a statement number specified at line XX

The statement number specified by the ON GOSUB or on GO TO statement does not exist in the program.

?MINC-F-GOSUB fails; 20 subroutines already active at line XX

You have tried to nest subroutines more than 20 levels deep.

?MINC-F-Reached RETURN without executing a GOSUB statement at line XX

Related

GO TO
GOSUB
Branching
Nesting

References

Book 2, Chapters 5 and 9.

Examples

The following ON/GO TO statement means "if the value of the expression is n, transfer control to the nth statement number in the list."

```
ON X1 GO TO 32767, 200, 300, 100
```

If the control variable X1 had the value 3, MINC would transfer control to statement number 300 because 300 is the third statement number in the statement number list.

OPEN

Purpose

Use an OPEN statement in your program to assign a channel number to an input file or output file that is stored on a diskette. You can open a channel in order to use an existing sequential data file or virtual array file or to create new sequential or virtual array files.

MINC provides 12 file channels for your use, numbered 1 through 12. All 12 file channels can be in use at the same time. Channel 0 is always reserved for the terminal. The OPEN statement assigns a file channel to a file. You can open a file at any point in your program, but you must open a file before MINC can execute any statement that requires the channel number. The following statements require channel numbers.

INPUT # channel, variable-list

LINPUT # channel, variable-list

PRINT # channel, USING description, print-list

IF END # channel THEN statement

Any reference to a virtual array file element (other than in the DIM statement which describes the virtual array file).

Forms

OPEN filespec FOR INPUT AS FILE # channel
DOUBLEBUF, FILESIZE value
FOR OUTPUT

Component	*Component Type*	*Component Value*	*Default Condition*
OPEN	statement	none	required component
filespec	string expression	dev:name.typ	dev: SY0: name required .typ .DAT
FOR INPUT FOR OUTPUT	statement component	none	input and output (virtual array files only)
AS FILE	statement component	none	required component
#	statement component	none	optional
channel	numeric expression	1 to 12	required component
DOUBLE BUF	statement component	none	normal file transfer
FILESIZE	statement component	none	normal space allocation
value	numeric expression	0 to available blocks on volume	paired with FILESIZE

The statement components *DOUBLE BUF* and *FILESIZE value* specify advanced capabilities of the OPEN statement. You can use either singly, or both. If you use both phrases, separate them with a comma, as shown in the following example.

```
OPEN "REP" FOR INPUT AS FILE #6 DOUBLE BUF, FILESIZE 30
```

The phrase DOUBLE BUF must precede the FILESIZE phrase. There is no comma before DOUBLE BUF if it is in the OPEN statement, but there is always a comma between DOUBLE BUF and FILESIZE. If you omit DOUBLE BUF, you do not need a comma before FILESIZE.

NOTE

When you open a file, MINC uses 256 words of the workspace for each file you open. Thus, if you open three files at the same time, you reserve 3×256=768 words of the workspace. MINC uses 512 words of the workspace for each file opened with the DOUBLE BUF option.

Instructions

An OPEN statement has five components explained in the following paragraphs.

filespec The filespec argument is the specification for the existing file to read data from input transfers. For output transfers, OPEN creates a file with the name specified when none exists. If the file specified for output exists, MINC supersedes a sequential file and updates a virtual array file. The line printer device, LP:, is a valid filespec argument.

FOR INPUT/FOR OUTPUT When you open a sequential file, the phrase FOR INPUT opens the file for input only. The phrase FOR OUTPUT opens the file for output only. If you specify neither, MINC opens the file for input if the file already exists, or opens the file for output if the file does not currently exist. You cannot open a sequential file for both input and output.

When you open a virtual array file, you can open it for input, for output, or for both. For both types of transfers, omit the FOR INPUT and FOR OUTPUT components.

When you want to use the line printer, remember that it is an output-only device. Include the FOR OUTPUT component in the OPEN statement.

 # channel MINC has thirteen file channels numbered 0 through 12. OPEN can assign one file at a time to each channel. You can assign file channels in any order.

File channel 0 is the channel permanently assigned to the MINC terminal. You cannot use channel 0 to read data from a diskette file or send data to a diskette file, and you cannot reference channel 0 in an OPEN statement.

If you want to change the direction of transfer or use a different file on a channel that is open, use a CLOSE statement to close the file and the channel. Use another OPEN statement to open the channel for your new work.

DOUBLE BUF Buffering is the technical term for describing how MINC manages the transfer between the file and the workspace. Using the DOUBLE BUF component in an OPEN statement can result in faster file processing. However, with DOUBLE BUF, MINC assigns twice as much workspace to the file and the program might exceed the workspace when it runs. It is appropriate to use DOUBLE BUF with virtual array files only when access to the elements is sequential or very clustered.

FILESIZE value The FILESIZE component allows you to specify the size (in blocks) of the output file. This capability can be useful when you are creating files on a relatively full volume.

The OPEN statement uses the normal method used by the system utilities for selecting file locations on a volume. It first locates the largest and second largest unused areas. Then it reserves either half of the largest area or all of the second largest area, whichever is greater.

Using the FILESIZE component, you can select one of three procedures for selecting the file location. The following table shows the meanings of different FILESIZE arguments.

Value	*Procedure*
-1	Reserve the largest unused area.
0	Use the normal procedure.
>0	Reserve the number of blocks specified by the value. Check the volume's directory to ensure that it contains unused areas at least as large as required.

Restrictions

MINC permanently stores output files on the volume only when a CLOSE statement executes. One safe procedure is to close each open file as you finish using it. This avoids losing output files if the program halts unexpectedly.

When MINC executes a STOP statement or when it stops because of a programming error, it does not close any files that are open. However, MINC does close all files when it executes an END statement. If MINC displays an error message or the message "STOPPING AT LINE stmt#", you can use an immediate mode CLOSE statement to protect files you have opened for output. When MINC displays READY after you have run a program, it has executed an END statement.

MINC uses 256 words of the workspace for each file that is open. If your program needs many variables and arrays, have as few files as possible open simultaneously. MINC uses 512 words of the workspace for each file whose OPEN statement contained the DOUBLE BUF component. Operations involving files are significantly slower than operations involving only the workspace.

Errors

?MINC-F-OPEN statement for file channel prohibits transfer at line XX

You tried to input from a file opened for output, or to print to a file open for input.

?MON-F-Trap to X XXXXX

The volume is not initialized.

The device does not exist; for example SY2: as a typographical error.

In both of these cases, the workspace is scratched.

?MINC-F-OPEN fails; no suitable free space on volume for file at line XX

The file that is being opened for output is larger than the available space on the volume.

?MINC-F-Specified or default volume does not have file named

The file that you are opening for input does not exist on the volume.

?MINC-F-OPEN fails; workspace too full for another channel at line XX

You have tried to open more channels than there is room in the workspace. Remember that each channel requires 256 words of the workspace (512 for DOUBLE BUF).

?MINC-F-Need OPEN statement for file channel at line XX

You forgot an OPEN statement but used the channel number in some statement such as LINPUT #.

?MINC-F-OPEN fails; file channel already open at line XX

You tried to open the same channel more than once without closing it in between.

?MINC-F-Value of FILESIZE expression too large or less than -1 at line XX

?MINC-F-File space allocated on volume is too small at line XX

The amount of space allocated by FILESIZE is inadequate.

The volume does not have enough available space. If you have been using the default file allocation, use the FILESIZE option to increase the space your program can use.

Related

CLOSE
COLLECT
File Allocation
IF END #
INPUT #
LINPUT #
PRINT #
STOP
END

References

Book 2, Chapter 11.

Examples

The following short program prints out "This is a test of the line printer" on the line printer.

```
10 OPEN 'LP:' FOR OUTPUT AS FILE 1
20 PRINT #1, 'This is a test of the line printer'
30 CLOSE
40 END
```

OVERLAY

Purpose

When you have divided a program into segments that can be overlaid, use the OVERLAY statement to merge a segment with the statements that are currently in your workspace.

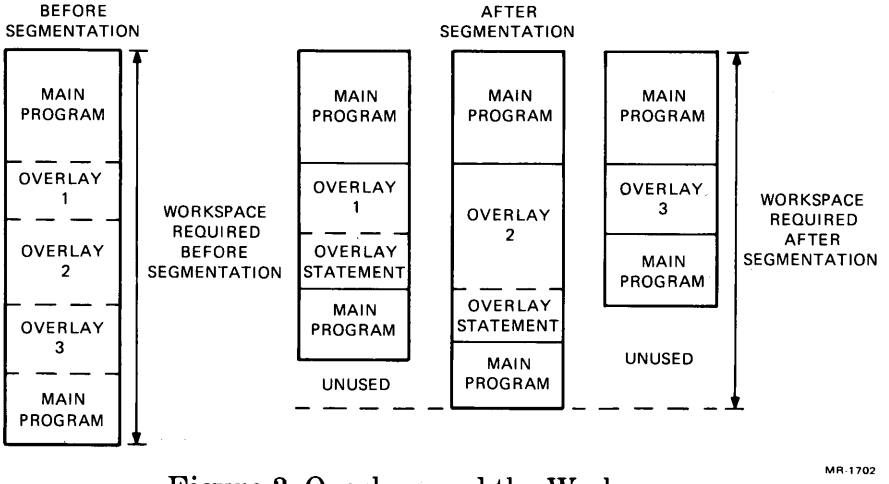

Figure 3. Overlays and the Workspace

Forms

OVERLAY 'filespec' LINE stmt#

Component	*Component Type*	*Component Value*	*Default Condition*
OVERLAY	statement	none	required component
filespec	string expression	dev:name.typ	dev: SY0: name required .typ .BAS
LINE	statement component	none	next statement
stmt#	numeric expression	1 to 32,767	paired with LINE

The LINE stmt# argument is optional. If LINE and the statement number are present, they represent the statement number at which MINC starts execution after the overlay. If you omit LINE and the number, MINC starts execution at the next sequential statement number after the OVERLAY statement.

Instructions

When MINC reads a line of the program segment from the file, it merges the statement into the current program. If a line with the same statement number already exists, MINC deletes the existing statement and replaces it with the statement from the new program segment. During this process, all variables and

arrays retain their current values, and all open files remain open. When all lines of the program segment in the file are read into the workspace and merged with the original program lines, MINC continues execution of the merged program at either the statement after the OVERLAY statement or the statement specified by the LINE stmt# component.

To segment a program with the OVERLAY statement, break the program into one main program and several overlay segments.

The total workspace required by a program segmented with the OVERLAY statement is the size of the main program plus the size of the largest overlay.

You must ensure that all line numbers in an overlay segment are repeated in each subsequent segment. Otherwise, parts of the previous segment remain in the workspace.

Unless its DEF statement is replaced by the overlay segment, user-defined functions are not affected by the overlay.

Note that if you enter the OVERLAY statement on a multistatement line, MINC ignores the rest of the line.

Restrictions

You cannot specify a compiled file with the OVERLAY statement.

Errors

?MINC-F-Specified or default volume does not have file named

The program file specified in the OVERLAY statement does not exist on the volume.

?MINC-F-Program too large; workspace overfills at line XX

The overlay segment creates a program that is too large for the workspace.

You tried to overlay a compiled program.

?MINC-F-Program does not have a statement number specified at line XX

The statement number specified in the LINE phrase does not exist in the overlay segment.

?MINC-F-END statement does not have highest number in program at line XX

The overlay caused the END statement not to be the last in the program.

Related

APPEND
CHAIN

References

Book 2, Chapter 14.

Examples

None included.

PI

Purpose

The PI function takes on the value π.

Forms

PI

Component	*Component Type*	*Component Value*	*Default Condition*
PI	function name/real	3.14159	required component

Instructions

Use PI in any expression where the value of the mathematical constant π is required.

Restrictions

Numeric precision — π is an irrational number that cannot be represented exactly.

Errors

None applicable.

Related

Numeric precision.

References

Book 2, Chapter 2.
Book 3, Numeric Precision.

Examples

```
Example   PRINT PI
Result      3.14159

Example   A=COS(PI)
          PRINT A
Result    -1
```

POS

Purpose

The POS function scans a string for the occurrence of a search model. If the string contains the search model, POS takes on the value of the location in the string of the first character of the model.

Forms

POS(string,search-model,start-position)

Component	*Component Type*	*Component Value*	*Default Condition*
POS	function name/numeric	0 to LEN(string)	required component
string	string expression	any valid string	required component
search-model	string expression	any valid string	required component
start-position	numeric expression	1 to LEN(string)	required component

Instructions

Use POS to determine whether or not a string contains a *search model* and to locate the position in the string of the first character in the search model. The first character in a string is at position 1, not position 0.

The search begins at the character position specified by the start-position argument. If the string contains the search model, POS takes on the character position of the first character of the search model. If the string does not contain the search model, POS takes on the value 0.

The following list defines the operation of POS under certain limiting conditions.

1. If the search model is null and the string contains characters, then POS takes on either the value of the start-position argument or the value LEN(string) + 1 (whichever is less).

2. If the string argument is null, then POS takes on the value 0.

3. If start-position argument has a value less than 1, then POS starts the search at the first character in the string. That is, POS operates as if the start-position argument had been 1.

4. If the start-position argument is greater than LEN(string) and the search model contains characters, then POS takes on the value 0.

5. The search fails and POS takes on the value 0 when the search-model is longer than the string to be searched.

Restrictions

The case of the characters in the strings does matter. The case of the characters in the search model must match exactly the case of the characters in the string for the search to succeed.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

One of the arguments is of the wrong data type.

Related

The LEN and SEG$ functions are related string processing functions. The LEN function determines the number of characters in a string. The SEG$ function extracts a segment string from a string.

References

Book 2, Chapter 8.

Examples

```
Example  PRINT POS('calculator','a',1)
Result     2

Example  A=POS('calculator','A',1)
         PRINT A
Result     0

Example  PRINT POS('calculator','a',2)
Result     2

Example  PRINT POS('calculator','a',3)
Result     7

Example  PRINT POS('calculator','w',1)
Result     0
```

PRINT

Purpose

Use a PRINT statement to display text and values, to send data to a diskette file, and to print data directly from a program to line printer. By composing PRINT statements in different ways and using different programming techniques, you can use your MINC system to produce a wide variety of line, screen, and file designs. The most common specific uses for PRINT statements are as follows.

- Create blank lines
- Prompt program users for file names and initial values
- Print messages about how a program is running
- Transfer string or numeric values to a diskette file
- Create tables and charts for screen display

NOTE

The PRINT USING statement is another form of the PRINT statement. Because the details of the PRINT USING form are so numerous, they have been moved to the section entitled PRINT USING.

Forms

PRINT #channel,USING description,print-list

Component	*Component Type*	*Component Value*	*Default Condition*
PRINT	statement	none	required component
#	statement component	none	paired with channel
channel	numeric expression	0 to 12	terminal screen (0)
USING	statement component	none	paired with description
description	string expression	# , . * $ ^ - 'L R C E	normal print format
print-list	list	see list element form	blank print line
Form of list element:			
item	string or numeric expression	any valid value	

Instructions

A PRINT statement has the three components described in the following paragraphs.

#channel You can direct program output to any file channel open for output except a virtual array file. Use the channel you

assigned to the file in the OPEN statement. Whenever you specify a channel number in a PRINT statement, put a number sign (#) before the channel number and a comma after the number.

If you do not specify a channel number in a PRINT statement, MINC uses the default #0 and directs the PRINT statement's output to your terminal.

USING description See the section entitled PRINT USING.

print-list The print-list argument can be null or any valid list of valid expressions separated by commas or semicolons. A null print list displays a blank line or concludes a display line started by a hanging PRINT statement (one ending with a comma or semicolon).

MINC requires separators between items in the print list. With a PRINT USING format, it does not matter whether the separators are commas or semicolons because the spacing is controlled by the format description, not by the normal conventions for PRINT.

MINC considers the terminal screen to have five 14-column zones. If you have your terminal set at 132 columns, you must execute the TTYSET system function to set the number of columns to 132 for a PRINT statement.

MINC considers the line printer to have nine 14-column zones and 132 available columns.

Restrictions

None included.

Errors

?MINC-F-Need OPEN statement for file channel at line XX

You forgot the OPEN statement.

?MINC-F-File space allocated on volume is too small at line XX

You are trying to put more information into the file than there is room. You can increase the default file size by using the FILESIZE option of the OPEN statement.

?MINC-F-Invalid PRINT USING format or syntax at line XX

Related

TAB function
PRINT USING
OPEN

References Book 2.

Examples

Example PRINT 2+3
Result 5

Example PRINT '2+3'
Result 2+3

PRINT USING

Purpose

The purpose of the PRINT USING form of the PRINT statement is to allow you to describe exactly what you want your output line of the PRINT statement to look like. Using the PRINT USING form of the PRINT statement, you can format your output lines.

Forms

PRINT #channel, USING description, print-list

Component	*Component Type*	*Component Value*	*Default Condition*
PRINT	statement	none	required component
#	statement component	none	paired with channel
channel	numeric expression	0 to 12	terminal screen (0)
USING	statement component	none	paired with description
description	string expression	# , . * $ ^ - 'L R C E	normal print format
print-list	list	see list element form	blank print line
Form of list element:			
item	string or numeric expression	any valid value	

Instructions

You can specify the format of the PRINT statement output precisely by including the USING component in the statement. The USING component has two parts, the keyword USING and a format description that PRINT uses when it executes the statement. (See the USING section and the section on PRINT USING in Book 2.) Enclose the format description in matching quotes and follow it with a comma. The following table summarizes the valid format description symbols and their principal meanings.

Symbol	*Meaning*
#	Reserve a space for a digit
.	Position of a decimal point
,	Separate thousands and hundreds with a comma
**	Fill with asterisks before first digit
$$	Print a dollar sign before first digit
^^^^	Print real number in E format
-	Follow negative number with minus sign

'	Start a string field
L	Left-justify string in field
R	Right-justify string in field
C	Center string in field
E	Expand string field if necessary

For the rest of the instructions for the PRINT USING form of the PRINT statement, see the PRINT statement.

Restrictions

To print more than one value on a line with PRINT USING format control, ensure that there is a format description for each item in the list. That is, the format description describes a line. Each time PRINT USING starts at the beginning of a format description, it starts a new line.

Fatal Error Conditions

The "?MINC-F-Invalid PRINT USING format or syntax" error message is produced if:

1. The format description is not a valid string expression.
2. There are no valid fields in the format description.
3. A string is printed in a numeric field.
4. A number is printed in a string field.
5. A negative number is printed in a floating dollar sign or asterisk field that does not specify a trailing minus.
6. The items in the list are separated by characters other than a comma or semicolon.

Nonfatal Error Conditions

Nonfatal error conditions, that is, error conditions that do not terminate the program, occur if:

1. A number does not fit in the field.
2. A string does not fit in the field.
3. A field contains an invalid combination of characters.

4. Text to be printed forms a valid field.

If a number is larger than the field allows, MINC prints a percent sign followed by the number in the standard PRINT format.

If a string is larger than any field other than an extended field, MINC truncates the string and does not print the excess characters.

If a field contains an invalid combination of characters, the first invalid character and all characters to its right are not recognized as part of the field. These characters may form another valid field or they may be considered text. If the invalid characters form a new valid field, this unintended field may cause a fatal error condition.

Errors

?MINC-F-Need OPEN statement for file channel at line XX

You forgot the OPEN statement.

?MINC-F-File space allocated on volume is too small at line XX

You are trying to put more information into the file than there is room. You can increase the default file size by using the FILESIZE option of the OPEN statement.

?MINC-F-Invalid PRINT USING format or syntax at line XX

Related

PRINT
OPEN

References

Book 2, Chapter 13.

Examples

The following examples show invalid combinations of characters in numeric fields.

Invalid Combinations. In the following example, two dollar signs are combined with two asterisks. $$ is a complete field and **##.## forms a second valid field. $5 is printed by $$ and **16.30 is printed by **##.##.

```
10 PRINT USING "$$**##.##",5.41, 16.30

RUNNH

$5**16.30
```

```
READY
```

The same invalid combination appears in the following example, but the next list item is a string. MINC produces the fatal error message after trying to print the string "ABC" in the unintended numeric field **##.##.

```
10 PRINT USING "$$**##.## 'LLL",5.41, "ABC"

RUNNH

$5
?MINC-F-Invalid PRINT USING format or syntax at line 10

READY
```

In the next example, the numeric field has only three, not four, carets.

The number does not fit in the field ##.#; a percent sign and the number are printed followed by three carets.

```
10 PRINT USING "##.#^^^",5.43000E+09

RUNNH

% 5.43000E+09

READY
```

In the following example, two letters cannot be combined in one field. The string constant EEE is printed.

```
10 PRINT USING "'LLEEE","VWXYZ"

RUNNH

VWXEEE

READY
```

Attempting to print characters as text produces error when the characters form a valid field. For example:

```
10 PRINT USING "THERE ARE ### # ## PENNY NAILS",123,4,16,6
```

is an attempt to print

```
THERE ARE 123 # 4 PENNY NAILS
THERE ARE 16 # 6 PENNY NAILS
```

but instead produces

```
RUNNH

THERE ARE 123 4 16 PENNY NAILS
THERE ARE 6

READY
```

To correctly print characters that would form a valid format description field, you must use a string field in the format description and place the characters as a string constant in the list. For example:

```
10 B$="THERE ARE ### '## PENNY NAIL"
20 PRINT USING B$,123,'#',4,16,"#",6

RUNNH

THERE ARE 123 # 4 PENNY NAILS
THERE ARE 16 # 6 PENNY NAILS

READY
```

Note that the '## in line 10 represents two fields — a one-place string field (') and a two-place numeric field(##).

This method is also the only way to print a single or double quotation mark character with the PRINT USING statement. For example:

```
LISTNH
10 PRINT USING "HE SAID, 'I'M GOING ' ", ' " ', " ' ", ' " '

READY
RUNNH

HE SAID, "I'M GOING."

READY
```

Protected File Types

Purpose

MINC system utilities and routines are actually stored in files on the system diskettes. To spare you the inconvenience of having all the system file names in your directory and to safeguard the system files against accidental deletion, the MINC system files are *protected.* That is, the file types for system files are invalid in all MINC commands and statements requiring a file specification as an argument.

Forms

The protected file types are shown in the following list.

.BAD

.COM

.SAV

.SYS

Instructions

Do not attempt to use protected file types in file specifications for your own files. MINC does not let you perform any operations on protected files.

Restrictions

You do see the protected file types in several situations. If a volume develops a bad block in a system file, the name of that file appears in the report from the VERIFY command. If you obtain a directory of the Master diskette, several files with protected types appear in the report.

Errors

?MINC-F-Use another file type; SYS, SAVE, COM and BAD are protected

Related

File Specifications

References

File Specifications

Examples

None included.

RANDOMIZE

Purpose

Normally, MINC produces the same sequence of pseudo-random numbers each time you run or chain to a program that uses the RND function. (The section on RND covers the function in detail.) If your program has a RANDOMIZE statement, MINC produces a different sequence of pseudo-random numbers each time the program runs.

The RND function simulates a random number generator by choosing a number from a fixed sequence of pseudo-random numbers. Without a RANDOMIZE statement, RND always chooses values from the sequence starting from the same point in the sequence. After MINC executes a RANDOMIZE statement, RND selects its next value from a different point in the pseudo-random sequence than it would have otherwise.

When MINC has not executed a RANDOMIZE statement, the value of a pseudo-random number depends on the last pseudo-random number used, a fixed calculation that takes no other external factors into consideration. That is, the complete sequence is fixed (hence, pseudo-random). When MINC executes a RANDOMIZE statement, the next value of RND depends on a complicated calculation which uses unpredictable parameters (the system clock value, the number of characters you have typed, the number of characters MINC has displayed on your terminal).

Forms

RANDOMIZE

Component	*Component Type*	*Component Value*	*Default Condition*
RANDOMIZE	statement	none	required component

Instructions

Develop and thoroughly test your program without a RANDOMIZE statement first. When you are satisfied that your program handles the standard set of pseudo-random values properly, add the RANDOMIZE statement.

Each time you run a program that has a RANDOMIZE statement, the set of pseudo-random numbers from the RND function is the same for statements that precede the RANDOMIZE statement. After MINC executes the RANDOMIZE statement,

the sets of pseudo-random numbers the RND function produces are different each time you run the program.

For example, when your RANDOMIZE statement is in an overlay or subroutine and your main program uses the RND function both before and after MINC executes the subroutine or overlay,

- the set of RND values in the main program is always the same before MINC executes the subroutine or overlay statements;
- the set of RND values in the main program is always different after MINC executes the subroutine or overlay statements.

When you chain programs that use the RND function, each program must include a RANDOMIZE statement if you want unpredictable RND values throughout the chain.

The only way to cancel the effect of a RANDOMIZE statement is to erase the workspace with a CHAIN statement or an OLD, NEW, SCR, or RUN command.

Restrictions None included.

Errors There are no error messages associated with this function.

Related
RND function
Erasing (OLD, NEW, SCR, RUN, CHAIN)
OVERLAY

References Book 2, Chapter 8.

Examples Book 1, Permutation Demonstration.

RCTRLC

Purpose

The RCTRLC system function disables normal CTRL/C operation. If you need to terminate a program externally while CTRL/C is disabled, you must restart the system manually.

Forms

variable=RCTRLC

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric variable	unknown	required component
=	none	none	required component
RCTRLC	system function	none	required component

Instructions

Use RCTRLC to disable normal CTRL/C operation.

The variable has some unknown value after you use the RCTRLC system function. You can use RCTRLC to protect a critical program against keyboard intervention. However, after using RCTRLC you can halt the program only with manual methods like turning off the power or using the BREAK key to restart. With either of these manual methods, you lose the contents of the workspace (which is normally preserved during a CTRL/C operation).

Restrictions

The RCTRLC function does not work in immediate mode. The system utility which displays READY reenables CTRL/C operation.

Errors

None included.

Related

CTRLC reenables normal operation of CTRL/C.

SYS(6) records whether CTRL/C has been pressed while CTRL/C is disabled. Therefore, when you reenable CTRL/C operation (with the CTRLC function), you can determine whether anyone attempted to halt the program while CTRL/C was disabled.

References

None.

Examples

None included.

RCTRLO

Purpose

The RCTRLO system function ensures that output to the screen is displayed. That is, RCTRLO cancels the effect of the most recent CTRL/O combination which inhibited screen output.

Forms

Variable=RCTRLO

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric variable	unknown	required component
=	none	none	required component
RCTRLO	none	none	required component

Instructions

Use RCTRLO to cancel any previous CTRL/O combination. The RCTRLO function does not require that CTRL/O be pressed previously. That is, it operates properly whether or not it was preceded by CTRL/O.

Restrictions

None included.

Errors

None included.

Related

There is no related CTRLO function.

References

Book 3, CTRL Operations.

Examples

None included.

READ

Purpose

Use a READ statement whenever you want to read a value or set of values from a DATA statement. Usually a READ statement is used to assign a value to a variable that remains constant throughout an entire run of a program but might change from run to run.

Forms

READ data-list

Component	*Component Type*	*Component Value*	*Default Condition*
READ	statement	none	required component
data-list	list	see list element form	required component
Form for list elements:			
item	string or numeric variable name	READ assigns value	

Instructions

Complete each READ statement with a list of string and numeric variables that are in the same order as string and numeric values in the corresponding DATA statements.

Each time MINC executes a READ statement, READ assigns the next DATA statement value to the next variable in the data-list. You can use a RESET (RESTORE) statement to return MINC to the first DATA statement value in your current program.

Restrictions

You cannot use a READ statement to acquire data values from a file.

Errors

?MINC-F-Too few values for INPUT or READ variables at line XX

There were not enough values in a DATA statement for the number of READ variables.

?MINC-F-DATA value or value from file does not match variable at line XX

The READ statement is expecting a numeric variable, but the corresponding value in the DATA statement is a string.

The DATA statement is not the last statement in a multiple statement line (which causes MINC to interpret the last value as a string value).

Related

DATA
INPUT
RESET
Assignment statement

References

Book 2, Chapter 12.

Examples

See Book 2.

REM

Purpose

Use REM statements to include comments about individual statements and groups of statements in your program.

Forms

REM

Component	*Component Type*	*Component Value*	*Default Condition*
REM	statement	none	required component
text	up to 70 characters	all except \	null remark

Instructions

MINC accepts any ASCII character in a remark. A remark is terminated by the \ (backslash) character or by the RETURN key.

MINC does not execute REM statements. When it identifies the keyword REM at the beginning of a statement, it ignores all other characters until it encounters a terminator.

Informative remarks are useful both to a program's author and to other people who have to read a program. Remarks that explain the roles of different variables, the purposes of subroutines and functions, and the program algorithm can be particularly useful.

Restrictions

The left parenthesis character, (, deletes all spaces following it in the remark. The left bracket character, [, does not have this effect and can be used instead of the left parenthesis.

Although MINC does not execute REM statements, they do occupy workspace. A program that fits in your workspace may nevertheless be too large when a RUN command allocates space to variables and arrays. One solution to the problem is to remove REM statements. However, you can also divide your program into parts and use CHAIN and OVERLAY statements to execute the parts separately. Using overlays and chaining techniques might sometimes be the best solution, particularly when your REM statements clarify a large, complex program.

REM statements can have a special use in overlaid programs. One way to ensure that a small overlay completely replaces a larger group of statements is to use the same set of statement numbers in each overlay and complete the superfluous state-

ments with the keyword REM. The dummy REM statements use much less of your workspace than most of the program statements they replace when MINC executes an OVERLAY statement. Since MINC does not execute them, they are probably more efficient than branching around superfluous statements.

Errors

There are no errors.

Related

Comments

References

Book 2, Chapter 3.

Examples

Example

```
10 REM Try out (parentheses) in a remark.
20 REM Try out [brackets]  in a remark.
30 REM Method for using (..parentheses) in a
remark.
```

Result

```
10 REM Try out (parentheses) in a remark.
20 REM Try out [brackets]  in a remark.
30 REM Method for using (..parentheses) in a
remark.
```

RENAME

Purpose

Use the RENAME command to change the name of the workspace whenever MINC displays READY.

The RENAME command does not change your program in any way, but it does assign the new name you specify to the workspace.

You cannot use the RENAME command to change the name of a file on a diskette. That is done by the NAME statement.

Forms

RENAME filename

Component	*Component Type*	*Component Value*	*Default Condition*
RENAME	command	none	required component
filename	characters	name	NONAME

Instructions

When MINC is READY, complete the command with a new file name.

If your workspace is empty, the RENAME is equivalent to a NEW command. However, if the workspace is not empty, the NEW command erases the contents whereas the RENAME command does not.

Restrictions

The RENAME command lets you choose the name only; it does not recognize a device or an extension. Although you can enter a complete file specification, RENAME ignores all but the name portion of the specification. If you later execute SAVE or REPLACE, your program is saved on device SY0: with the .BAS extension.

Errors

?MINC-F-Invalid file name

There are invalid characters in the file name. In this case, MINC defaults the name to NONAME.

Related

NEW

References

None.

Examples

Example	RENAME SINES
Result	The workspace name is now SINES.BAS. The contents of the workspace are unchanged.

REPLACE

Purpose

The REPLACE command erases a specified file from a volume and stores the workspace program on the same volume under the same name as the erased file.

Forms

REPLACE filespec

Component	*Component Type*	*Component Value*	*Default Condition*
REPLACE	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name workspace
name			.typ .BAS

Instructions

If you complete the command with a file name, MINC executes it directly.

If you do not specify a file name in the REPLACE command, MINC replaces the file with the same name as the current workspace name.

After MINC has replaced a file with a copy of your current program, you cannot recover the original file.

Two common mistakes with the REPLACE command are to mistype the file name or specify the wrong diskette volume in the file name. In either case, you can check the name you typed by using a LIST command which displays the current program name. Then use the DIR command to list the names of files on the diskettes you are using, if necessary. Use another REPLACE command or a SAVE command to store your program when you have decided about the proper name and volume. If you want to change the current program name back to the original name, use a RENAME command.

Restrictions

None included.

Errors

?MINC-F-Specified or default volume does not have file named

The file does not exist on the specified volume. Use the SAVE command instead of REPLACE.

?MINC-F-Invalid file name

The file name that you typed for the REPLACE command has characters that are invalid for a file name.

?MINC-F-OPEN fails; no suitable free space on volume for file

There is not enough room on the volume to replace the program.

Related

SAVE The SAVE and REPLACE commands both store your current program on a diskette. However, the SAVE command cannot erase any existing file. MINC prevents you from inadvertently erasing a file in this case and displays the message:

?MINC-F-File name in use; REPLACE or change name or volume

References

Book 2, Chapter 4.

Examples

See Book 2.

RESEQ

Purpose

When MINC displays READY, use the RESEQ command to change the statement numbers in your current program. You can change all statement numbers, certain groups of numbers, or just one number. In all cases, MINC identifies and changes every occurrence of each statement number you specify except for statement numbers that are within the comment portion of REM statements. For example, when you use a RESEQ command to change a statement number you have used in a GOSUB statement, MINC makes the change in the GOSUB statement as well as in the target statement.

The RESEQ command changes statement numbers only in your current program. When you change the statement numbers in one of the programs in a chain or in one part of a program you have divided into overlays, check the other files carefully. For example, the general technique of getting a file with OLD, changing its statement numbers with RESEQ, and storing its new form with REPLACE is a systematic way to revise all programs in a chain and all overlays.

Forms

RESEQ newstart, oldstart - oldfinish, increment

Component	*Component Type*	*Component Value*	*Default Condition*
RESEQ	command	none	required component
newstart	numeric literal	0 to 32,767	the increment value
,	statement component	none	paired with following arguments
oldstart	numeric literal	0 to 32,767	first program statement
-	statement component	none	paired with oldstart and oldfinish
oldfinish	numeric literal	1 to 32,767	last program statement
,	statement component	none	paired with increment
increment	numeric literal	1 to 32,767	10

Instructions

The full form of a RESEQ command has three components. You can specify the range of current statement numbers MINC is to change. You can specify the interval between new statement numbers MINC assigns. You can specify the first value MINC is to use for the new statement numbers it assigns. Specific instructions for each component follow.

Your program is always in statement number order in your workspace. If your RESEQ command would change the order of program statements at all, MINC displays the following message and does not execute the command.

```
?MINC-F-RESEQ has an invalid statement number or interval
```

MINC requires commas between the different components of a RESEQ command. When you study the examples that follow, note carefully how commas are used when you omit the start value or range.

newstart If you specify a starting value for the new statement numbers MINC is to assign, oldstart (the first statement number changed) becomes newstart.

If you omit the starting value, MINC uses the increment argument to calculate the first statement number in the changed set.

oldstart - oldfinish If you specify a range of current statement numbers, MINC changes only the current statement numbers that are in that range. Use a hyphen to separate the first and last statement numbers in the range.

If you omit the first statement number in the range, MINC assumes that oldstart is the first statement in the program. If you omit the last statement number in the range, MINC assumes that oldfinish is the last statement number in the program. If you omit either end of the range, you must include the hyphen. However, if you omit the range entirely (thus resequencing the whole program), you can omit the hyphen too.

increment If you specify an interval, MINC adds the increment to the preceding statement number to obtain each new statement number. The increment must be a whole number.

If you omit the increment, MINC uses a default increment of 10.

When you renumber the statements in a large program, you might inadvertently be asking MINC to try to create statement numbers larger than 32,767. In this case, MINC prints out the following error message.

```
?MINC-F-RESEQ has an invalid statement number or interval
```

Restrictions

After using APPEND to add another part of a program to your current program, you can use a RESEQ command to consolidate all of the statement numbers into one continuous group.

If you issue a RESEQ command that is invalid, MINC does not display an error message; instead, it just does not do anything.

Several lab module routines have a subroutine statement number as an argument. The RESEQ command does not adjust these arguments to reflect resequenced statement numbers. See Book 6.

Errors

?MINC-F-RESEQ has an invalid statement number or interval

The RESEQ command tries to create a line number that was larger than 32,767.

Related

None.

References

Book 2, Chapters 4, 5, and 9.
Book 6, Service Subroutines.

Examples

Example RESEQ 100

Result Renumbers all program statements using 10 as the increment. The first statement number in the renumbered program is 100.

Example RESEQ 100, 105-205

Result Renumbers statements 105 through 205 (inclusive) with an increment of 10, starting by changing statement 105 to statement 100.

Example RESEQ , -500

Result Renumbers all statements from the beginning of the program through statement 500 in increments of 10, starting the new sequence with 10.

Example RESEQ , 105-205

Result Renumbers statements 105 through 205 (inclusive) with an increment of 10. If we assume that the statement immediately preceding 105 is 100, then it starts by changing statement 105 to 110 (that is, 100 + 10).

Example RESEQ , 500-

Result Renumbers all statements from statement 500 to the end of the program. If we assume that the statement immediately preceding 500 is 200, then it starts by changing statement 500 to statement 210 (200 + 10).

Example RESEQ 250, 0-500, 5

Result Renumbers all statements through statement 500 using an increment of 5. The first new statement number is 250.

Example RESEQ 250, , 20

Result Renumbers all statements in the workspace using an increment of 20. The first new statement number is 250.

Example RESEQ ,,20

Result Renumbers all statements in the workspace using an increment of 20. The first new statement number is 20.

RESTORE/RESET

Purpose

Use the RESTORE statement to return MINC to the first data value in a sequential data file or to the beginning of the first DATA statement in your program. For historical reasons, MINC also includes the RESET statement, but RESTORE and RESET are exactly equivalent.

Each RESTORE statement affects one set of data, either the set in your DATA statements or a set in a sequential data file. After MINC executes a RESET statement that specifies a sequential data file, later INPUT and LINPUT statements that use that file read values from the beginning of the file. After MINC executes a RESET statement that has no file number, later READ statements read values from the beginning of the values in DATA statements.

Forms

RESTORE # channel
RESET # channel

Component	*Component Type*	*Component Value*	*Default Condition*
RESTORE/RESET	statement	none	required component
#	statement component	none	paired with channel
channel	numeric expression	1 to 12	refers to DATA

Instructions

Complete the RESTORE statement with a file number if you want to process the file's data again from the beginning. Use the file number that you used in the corresponding OPEN, INPUT, or LINPUT statements.

Omit the file number if you want to process DATA statement values again from the beginning.

To reset a file channel, the corresponding file must be open or MINC will display the following message.

```
?MINC-F-Need OPEN statement for file channel at line 10
```

Restrictions

None included.

Errors

```
?MINC-F-Need OPEN statement for file channel at line XX
```

You used a RESTORE # statement without opening a file on that channel.

?MINC-F-OPEN statement for file channel prohibits transfer at line XX

You tried to restore a file that is open for output.

Related

OPEN
INPUT
LINPUT
READ
DATA

References

Book 2, Chapters 11, 12.

Examples

See Book 2.

RESTART

Purpose

The RESTART command initiates the same sequence of events as turning on the power. Use the RESTART command whenever you change the system volume in SY0:. The RESTART command ensures that MINC recognizes the differences between the previous volume in SY0: and the new volume in SY0:.

Forms

RESTART

Component	*Component Type*	*Component Value*	*Default Condition*
RESTART	command	none	required component

Instructions

Always type the RESTART command whenever you change the system volume in SY0:.

Restrictions

The RESTART statement initiates the system start procedure. You lose all workspace contents.

Errors

None included.

Related

Start Procedures
BYE

References

Book 2.

Examples

Example RESTART

Result The terminal screen flashes, MINC pauses, and then requests that you enter the date and time.

RETURN

Purpose

Use RETURN statements within a subroutine at the point you want MINC to transfer back to the part of your program that called the subroutine. When MINC executes a RETURN statement, it terminates the subroutine processing. The statement that MINC executes after RETURN is the statement that follows the last GOSUB it executed.

A subroutine can contain more than one RETURN statement, and RETURN is a valid statement within an IF statement. For example, you can use one RETURN statement to transfer control when a value is outside a valid range and a different RETURN statement after statements that process a value inside the valid range.

Forms

RETURN

Component	*Component Type*	*Component Value*	*Default Condition*
RETURN	statement	none	required component

Instructions

The RETURN statement is only valid within a subroutine. Every subroutine must have at least one RETURN statement.

Restrictions

None included.

Errors

?MINC-F-Reached RETURN without executing a GOSUB statement at line XX

?MINC-F-GOSUB fails; 20 subroutines already active at line XX

You used a GO TO from a subroutine rather than a RETURN, and the program executed more than 20 GOSUB statements for that subroutine.

You have tried to nest subroutines more than 20 levels deep.

Related

GOSUB

References

Book 2, Chapter 8.

Examples

See Book 2.

RND

Purpose

The RND function takes on the value of a pseudo-random number "in the range" 0 to 1.

Forms

RND

Component	*Component Type*	*Component Value*	*Default Condition*
RND	function name/real	0 to 1	required component

Instructions

Use RND to generate a pseudo-random number in the range 0 to 1 with a normal distribution.

In most cases, the application requires a number in a range other than 0 to 1. Perform a linear transformation on the RND value to obtain a value in the required range. For example, if you require a whole number in the range 1 to 10, the following statement produces the number:

```
X=INT(RND*10+1)
```

In general, for an integer result, calculate the transformation using the width of the range required (W) and the offset from 0 of the lower end of the range (L).

```
number = INT(RND*W+L)
```

Check the calculation very carefully to be sure that you obtain numbers with the distribution you are expecting.

Develop and thoroughly test your program without a RANDOMIZE statement first. When you are satisfied that your program handles the standard set of pseudo-random values properly, add the RANDOMIZE statement.

Restrictions

The numbers are called pseudo-random because they are based on a mathematical calculation which always produces the same sequence of numbers. The important feature of the pseudo-random sequence is that it has a very long period. That is, the sequence itself repeats only after a large number of values. Thus, for all practical purposes, you can treat the RND result as if it were a value sampled with replacement from a uniform frequency distribution.

Errors

None.

The RND function does not produce an error message when you try to specify an argument; it ignores any argument except RND itself.

Related

The RANDOMIZE statement changes the point in the pseudo-random sequence from which RND draws its next value.

References

Book 2, Chapter 8.

Knuth, D. E., *The Art of Computer Programming.* Vol. 2. *Seminumerical Algorithms.* Reading, Mass: Addison-Wesley, 1969.

Examples

See Book 2.

Routines

Purpose

Your MINC system includes special routines that create graphic displays and manage data transfers to and from laboratory instruments. The routine names function as statement keywords in program statements. The arguments in a routine statement specify values needed by the routine and identify variables that will contain the results generated by the routine.

Forms

CALL routine-name(arguments)

Instructions

Each routine has a name that functions as a keyword in a program statement. The actual operation of the routine depends on the values of the arguments you specify in the statement. Refer to the appropriate book for instructions on valid routine statements. You can use many routines in immediate mode, as well as in program mode.

Restrictions

None included.

Errors

?MINC-F-CALL fails; workspace too full for parameters

Related

None included.

References

Book 2, Chapter 10.
Book 4.
Book 5.
Book 6.

Examples

None.

RUN/RUNNH

Purpose

When MINC displays READY, use the RUN command (or RUNNH) to execute a program. Each time you run a program, MINC performs the following operations.

- Scans each program statement
- Reserves part of your workspace for each array
- Notes each function you have defined in a DEF statement
- Sets each numeric variable and array element equal to zero
- Sets each string variable and array element equal to the null string (a null string is a string of length 0; it has no ASCII characters)
- Executes the program

Normally, the RUN command (or RUNNH) executes your current program. However, you can also execute a program you have stored in a diskette file directly by including the file name in the RUN command (or RUNNH). When you use a RUN command (or RUNNH) with a file name, MINC does the following processes before the steps in the list above.

- Erases the workspace
- Changes the workspace name to the new name
- Gets the new program from the file you specified

Therefore, if your current program is important, use the SAVE command or REPLACE command to store your current program before you use RUN to execute a stored program directly.

The only difference between RUN and RUNNH is that RUNNH does not display a heading.

Forms

RUNNH filespec

Component	*Component Type*	*Component Value*	*Default Condition*
RUNNH	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name workspace program .typ .BAC, then .BAS

Instructions

When you use the form of the RUN (or RUNNH) command that executes a program directly from a file, MINC first looks for the file with the .BAC extension (the compiled version). If there is no compiled version, then MINC executes the .BAS version.

If your program has stopped because of an error or because MINC executed a STOP statement, you can execute it from the beginning with the RUN command or have MINC continue from the statement number you specify in an immediate GO TO statement. The principal difference is that RUN and RUNNH cause MINC to set all program variables and array elements to the initial values of zeroes and null strings. An immediate GO TO statement causes MINC to continue executing your program with whatever values your program established by the time it stopped.

If you run a program directly from a file, MINC does not print the header line in either case (RUN or RUNNH).

The following form of the RUN command executes a stored program.

RUN name

MINC runs the program with the file specification SY0:name.BAC or SY0:name.BAS (depending on whether there is a compiled version).

If you type the entire file specification, then MINC runs the program in that file.

Restrictions

If you use the RUN filespec form of the RUN command, MINC displays no header. If you have a file named NHTEST and you enter RUN NHTEST, MINC tries to do a RUNNH command for the file TEST.

Errors

?MINC-F-Specified or default volume does not have file named

The file named in the RUN command does not exist on the volume.

?MINC-F-Syntax error; cannot translate the statement

There is a syntax error in the RUN command.

The first line of the file specified in the RUN command is unexecutable. This can happen if you edited the file with the keypad editor and made a mistake.

Related

SAVE
REPLACE

References

Book 2, Chapter 4.

Examples

Example	RUN
Result	MINC runs the current program in the workspace.

Example	RUN SINES
Result	MINC brings the SINES.BAS file into the workspace from SY0: and then runs it.

SAVE

Purpose

Whenever MINC is READY, use the SAVE command to store your current program in a diskette file. When MINC executes the SAVE command, it always creates a new file unless a file on the same diskette has the same name you want to use. When MINC finds a conflicting name, it displays the following message.

?MINC-F-File name in use; REPLACE or change name or volume

Forms

SAVE filespec

Component	*Component Type*	*Component Value*	*Default Condition*
SAVE	command	none	required component
filespec	characters	dev:name.typ.	dev: SY0: name workspace name .typ .BAS

Instructions

Complete the SAVE command with a valid file name when you want to store a program under a name that is different from your current program's name. If you are unsure whether the volume you want to use already has a file with the same name, use the DIR command to check the volume's directory.

Restrictions

None included.

Errors

?MINC-F-File name in use; REPLACE or change name or volume

?MON-F-Trap to X XXXXX

The volume specified in the SAVE command is uninitialized.

?MINC-F-Invalid file name

There are invalid characters in the file name specified in the SAVE command.

?MINC-F-I/O error; unable to check volume owner

You specified a device that does not exist in your file specification.

There is no diskette in the drive you specified.

?MINC-F-Use another file type; SYS, SAV, COM and BAD are protected

Related

REPLACE
OLD
NEW

References

Book 2, Chapter 4.

Examples

Example SAVE

Result MINC saves the current contents of the workspace in a file on SY0: with the current workspace name.

Example SAVE SINES

Result MINC saves the current contents of the workspace in a file on SY0: with the name SINES.BAS.

SCR

Purpose

Whenever MINC displays READY, use the SCR command to erase your current program entirely, to change your current program name to NONAME, to recover all of your workspace MINC assigned for the arrays you were using, and to cancel all variables and functions. After MINC executes the SCR command, you cannot recover the program that was in your workspace unless you stored it in a diskette file.

Forms

SCR

Component	*Component Type*	*Component Value*	*Default Condition*
SCR	command	none	required component

Instructions

If you want to use your current program again, use the SAVE command or the REPLACE command to store it on a diskette before you type SCR.

SCR abnormally terminates all open files — that is, if you have a file open when you use the SCR command, you will lose changes you made to the file.

Restrictions

None included.

Errors

There are no error messages associated with this command.

Related

CLEAR
NEW
OLD

References

Book 2, Chapters 2 and 3.

Examples

Example SCR

Result The workspace is now empty.

SEG$

Purpose

The SEG$ function takes on the value of a substring (segment) extracted from a string. The extracted substring is specified by its start and end character positions in the original string. The original string is unchanged.

Forms

SEG$(string,start-position,end-position)

Component	*Component Type*	*Component Value*	*Default Condition*
SEG$	function name/string	specified substring	required component
string	string expression	any valid string	required component
start-position	numeric expression	any numeric value	required component
end-position	numeric expression	any numeric value	required component

Instructions

The following list defines the operation of SEG$ in certain limiting conditions.

1. If the start-position argument is less than 1, SEG$ starts excerpting at character position 1. That is, SEG$ operates as if the start-position argument had been 1.

2. If start-position is greater than end-position or greater than LEN(string), SEG$ takes on the value of the null string.

3. If end-position is greater than LEN(string), SEG$ takes on the value of a segment string starting with the character at start-position and continuing to the end of the string. That is, SEG$ operates as if the end-position argument had been LEN(string).

4. If start-position equals end-position, then SEG$ takes on the value of the single character at that position.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

LEN, POS The LEN and POS functions are related string processing functions. The LEN function determines the number of characters in a string. The POS function determines the location of a search model in a string.

References

Book 2, Chapter 8.

Examples

Example A$=SEG$(CLK$,3,3)
PRINT A$

Result :

Example PRINT SEG$(DAT$,8,9)

Result 78

SGN

Purpose

The SGN function takes on a value corresponding to the sign of its argument.

Forms

SGN(number)

Component	*Component Type*	*Component Value*	*Default Condition*
SGN	numeric	-1, 0, or 1	required component
number	numeric expression	any valid value	required component

Instructions

Use SGN to determine the sign of the value of a numeric expression. The correspondence between SGN values and the sign of the expression is shown in the following table:

SGN value	*Argument value*
-1	The number argument is negative
0	The value of the number argument is 0
1	The number argument is positive

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

ABS The ABS function takes on the unsigned magnitude of its argument. The ABS and SGN functions are related by the following identity:

number = SGN(number)*ABS(number)

References

Book 2, Chapter 8.

Examples

```
Example   A=SGN(-3)
          PRINT A
Result    -1

Example   PRINT SGN(COS(PI/2))
Result     0
```

SIN

Purpose

The SIN function takes on the sine of its argument.

Forms

SIN(angle)

Component	*Component Type*	*Component Value*	*Default Condition*
SIN	real	-1 to +1	required component
angle	numeric expression	any angle, radians	required component

Instructions

Use SIN to determine the sine of an angle. The angle must be expressed in radians. The following formula converts an angle in degrees to an angle in radians:

radians = degrees*PI/180

Restrictions

The sine of PI and multiples of PI obtained with the SIN function are not the same as their mathematical definitions. The expression SIN(PI) has the value -1.87253E-07, which is effectively 0 for many purposes. However, the expression SIN(N*PI) often has the same value as the expression N*SIN(PI). As *N* increases, the absolute magnitude of the error increases and could quickly accumulate to affect the outcome of calculations with critical precision requirements due to the fact that π is an irrational number and cannot be exactly represented in a computer and the nature of the approximation function used to calculate the value of SIN.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

The COS and ATN functions provide the other trigonometric capabilities of MINC.

References

Book 2, Chapter 2.
Book 3, Numeric Precision.

Examples

See Book 2.

Special Keys

Purpose

Several keyboard keys not found on a typewriter keyboard have special functions in MINC.

Forms

None included.

Instructions

Not applicable.

Restrictions

The keys ↑, ↓, ←, →, operate only in the editor and in some of the graphic routines. When you are not using the editor, they are simply keys on the keyboard. You can use their character forms in programs.

DELETE Normally, the DELETE key works as if you pressed the sequence backspace, space, backspace. It deletes only one character internally regardless of the appearance of the character on the screen. That is, pressing the DELETE key once deletes the internal form of characters with more than one character in their screen form (control combinations and the TAB key).

NO SCROLL The NO SCROLL key that alternately stops the terminal screen from displaying information and starts the display. The NO SCROLL key has the same functions as the CTRL/S and CTRL/Q control characters.

TAB The TAB key sends up to eight spaces to the screen to simulate the action of typewriter tabulation. This key has no relation to the tab stops that you can set in SETUP mode.

Errors

None.

Related

CTRL operations

References

Book 2, Chapter 1.

Examples

None included.

SQR

Purpose

The SQR function takes on the value of the square root of its argument.

Forms

SQR(number)

Component	*Component Type*	*Component Value*	*Default Condition*
SQR	real	>=0	required component
number	numeric expression	>=0	required component

Instructions

Use SQR to determine the square root of a numeric value.

Restrictions

None included.

Errors

?MINC-W-Attempt to find square root of negative value

?MINC-F-Arguments in definition do not match function called at line XX

Related

None.

References

Book 2, Chapter 2.

Examples

Example
```
A=SQR(2)
PRINT A
```
Result
```
 1.41421
```

Example
```
PRINT SQR(SQR(16))
```
Result
```
 2
```

Start procedures

Purpose

The purpose of the start procedures is to make sure that your system starts up properly and that you do not inadvertently destroy one of your volumes.

Forms

Not applicable.

Instructions

To start your system when it is turned off, install a system volume in SY0: and turn the power on. MINC prompts you for the time and date.

To change system diskettes, remove the old system diskette, install the new system diskette, and type the RESTART command.

To change the volume in SY1:, remove the old volume, install the new volume in SY1:, and type the BYE command.

Restrictions

None included.

Errors

@

Diskette in SY0: is an uninitialized diskette.

?BOOT-F-No boot on volume

Diskette in SY0: is not a system diskette. It has been initialized.

?MON-F-System read failure halt

Diskette in SY0: probably has bad blocks in one or more of the system files. For a discussion of error recovery procedures, see "Error Recovery".

Related

RESTART
BYE
Error Recovery

References

None included.

Examples

None included.

STOP

Purpose

Each time MINC executes a STOP statement, it stops the program, reports the last statement number executed, and displays READY. At this point, MINC preserves both the program itself and the values of the variables and arrays the program was using. You can print them without changing them by using an immediate mode PRINT statement.

When you want MINC to continue executing your program, type a GO TO or GOSUB statement with the number of the statement MINC should start with.

Forms

STOP

Component	*Component Type*	*Component Value*	*Default Condition*
STOP	statement	none	required component

Instructions

The STOP statement does not close any sequential files or virtual array files that are open. Remember to use a CLOSE statement if you want to preserve any sequential files or virtual array files your program produced before stopping.

MINC accepts the STOP statement as part of a multistatement line, in an IF statement, and within FOR statement loops and subroutines. A program can contain as many STOP statements as you want.

The STOP statement is particularly useful while you are debugging a program. For example, by making MINC stop at appropriate points in your program, you can check intermediate values of variables, data your program has gotten with READ, INPUT, or LINPUT statements and the current values PRINT statements have put into sequential files and virtual array files that are open for output. When you are satisfied with the way a section of your program is working, you can use a DEL command to remove a STOP statement that you no longer need or a SUB command to change a STOP statement into a remark.

Restrictions

None included.

Errors	There are no error messages associated with this statement.
Related	END
References	Book 2, Chapter 3.
Examples	See Book 2.

STR$

Purpose

The STR$ function converts a numeric value to the corresponding string value. The STR$ function takes on the ASCII code for the number as its value.

Forms

STR$(number)

Component	*Component Type*	*Component Value*	*Default Condition*
STR$	string	string representation of number	required component
number	numeric expression	any valid value	required component

Instructions

Use STR$ to obtain the string representation of a numeric value.

Unlike the PRINT statement result, the string result of STR$ does not include an initial blank or a trailing blank. Therefore, you can use STR$ to print the value of a numeric expression without surrounding spaces.

Restrictions

The STR$ function converts numeric literals to the same form as the PRINT statement uses.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

VAL The VAL function converts a string expression to its numeric equivalent. The VAL function and the STR$ function reverse each other's effects, as shown by the following identity:

number = VAL(STR$(number))

**ASC, CHR$** The ASC and CHR$ conversion functions are not related to STR$. ASC converts a single character to its numeric ASCII code. CHR$ converts a numeric ASCII code to a single character string.

References

Book 2, Chapter 8.

Examples

Example A$=STR$(25.3)
PRINT A$
Result 25.3

Example PRINT STR$(12345678)
Result 1.23457E+07

Example PRINT STR$(6e1)
Result 60

Example PRINT VAL(STR$(6e1))
Result 60

SUB

Purpose

Whenever MINC displays READY, use the SUB command to change a single string of characters in one of your current program's statements. For example, the following command substitutes PRINT for the first occurrence of PIRNT in statement 200.

```
SUB 200 [PIRNT[PRINT[
```

The command is most useful when you need to make a short change to a long statement that you do not want to retype entirely.

You can also use the SUB command to move statements around in your program as shown in the last example below.

Forms

SUB stmt# [current-form [changed-form [which-occurrence

The left bracket ([ ) stands for any separator character that does not occur in either the current form or the changed form you describe.

Component	*Component Type*	*Component Value*	*Default Condition*
SUB	command	none	required component
stmt#	numeric literal	1 to 32,767	required component
current-form	characters	any program fragment	required component
changed-form	characters	any valid program fragment	required component
which-occurrence	numeric literal	1 to no. in statement	1

Instructions

The SUB command has three components. The following paragraphs describe the components in more detail.

MINC requires a valid statement number in each SUB command. The change you specify in a SUB command affects only the single statement you specify.

MINC requires a description of the current part of the statement you want to change and the final form you want that part to have. The current-form string must be entered exactly as it appears in the listed form of the statement (which is not necessarily the same as the form you typed; see LIST and COMPILE). If the

statement you are trying to change is not a valid program statement, then the differences between the internal form of the statement and what you think it is can cause problems for the SUB command. Use the LIST command to list the statement before you try to change it with SUB.

Choose a separator character that does not appear in the current or changed forms, and use that character both to separate and to enclose your description. In the example above, the separator is a left bracket ([ ). The left bracket ([ ) is a good character to use as a separator because it does not normally occur in BASIC statements, and it is not a shift character on the terminal keyboard.

You can use the SUB command to erase a string that is in a statement by specifying nothing as the final form. For example, the following command removes the first occurrence of (A+B) from statement number 500.

```
SUB 500 [(A+B)[[
```

Sometimes the string you want to change occurs more than once in a statement. You can specify which occurrence of the string to change using the occurrences argument. For example, the following command changes the third occurrence of B* to B/ in statement number 70. The first two occurrences of B* are not affected.

```
SUB 70 [B*[B/[3
```

When you do not specify which occurrence to change, SUB changes the first occurrence.

When you use the SUB command, MINC always lists the current form of the statement (with changes if the SUB command was valid and without changes if the SUB command was invalid). If you describe a string that does not occur in the statement you specify, MINC does not make any changes, and when MINC lists the statement you can see that it is unchanged.

Immediately after you use the SUB command to change the number of a statement, MINC creates a new statement and inserts it in numerical order in your current program. However, note that your original statement also remains in your program. In many cases, that will cause either a logic error and faulty results or a more serious program error that prevents MINC from executing your program. Remember to use a DEL command to remove the original statement, if you have used SUB to renumber a statement.

The most common way to renumber a statement in your program is to retype the statement with the appropriate statement number and use a DEL command to remove the original statement. Although you can also use the SUB command, the danger of forgetting to remove the original statement you changed is somewhat greater than if you simply retype it.

Restrictions

The SUB command does not check the occurrences argument for feasibility. If not enough occurrences exist in the line to make the change requested, then SUB does not change the line.

MINC depends on parentheses for several processes. When you use a SUB command to change the text of a REM statement that includes parentheses, MINC sometimes removes all spaces that occur after the left parenthesis. If this should happen, simply change the contents of the REM statement.

You cannot use the SUB command to make an immediate mode statement from a program statement.

Errors

?MINC-F-SUB creates an invalid statement or has a syntax error

The syntax of the changed statement is incorrect.

You tried to use the SUB command to remove the statement number.

Related

DEL

References

Book 2, Chapter 4.

Examples

Enter	20 fo to 50
LIST	20 foto50
Change	SUB 20 [foto [goto
Result	20 goto50
Inspect	LIST 20
Result	20 GO TO 50
Example	55 PRINT 'hello'
Enter	SUB 55 [55 [70
Result	70 PRINT 'hello'

Note that the statement number has been changed from 55 to 70. However, now both statements 55 and 70 exist in the program. If you do not want two copies of the statement, you must delete statement number 55.

SYS(1)

Purpose

The SYS(1) function reads a single character from the keyboard and takes on the numeric ASCII code value of the character.

Forms

variable=SYS(1)

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric	variable name	required component
=	none	none	required component
SYS(1)	function name/integer	0 to 127	required component

Instructions

Use SYS(1) to read characters as ASCII codes, one character at a time, from the keyboard. Each time the program executes SYS(1), SYS(1) is assigned the value of the next character in the most recent line of input from the keyboard. SYS(1) accepts its first character only after you have pressed the RETURN key. It does not accept any characters without the RETURN key.

The RETURN key itself actually generates two characters, carriage return (code 13) and linefeed (code 10). When you provide input to the SYS function by pressing only the RETURN key (with no preceding character) the first occurrence of SYS receives the value 13 and the next one receives the value 10.

In immediate mode, SYS(1) waits for the next input line from the keyboard.

Restrictions

The operation of CTRL/C is inhibited by SYS(1). When the program is awaiting input for SYS(1), it does not respond to CTRL/C from the keyboard. However, as soon as you press the RETURN key, the program responds to the CTRL/C by stopping and printing the messages STOP and READY.

Errors

None included.

Related

Use the INPUT statement and the ASC function for similar purposes.

References

None.

Examples

Example
```
10 PRINT 'enter a character, press RETURN'
20 A=SYS(1)
```

```
30 B=SYS(1)
40 C=SYS(1)
50 PRINT A,B,C
60 STOP
```

Result enter a character, press RETURN

m (RET)

109 13 10

Example w=sys(1)\x=sys(1)\y=sys(1)\z=sys(1)

Input abcd (RET)

Result PRINT w;x;y;z

97 98 99 100

Example w=sys(1)\x=sys(1)\y=sys(1)\z=sys(1)

Input abc (RET)

Result PRINT w;x;y;z

97 98 99 13

SYS(6)

Purpose

The SYS(6) function records whether or not two or more CTRL/Cs were entered on the keyboard while CTRL/C operation was disabled.

Forms

variable=SYS(6)

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric variable	variable name	required component
=	none	none	required component
SYS(6)	integer	0 or 1	required component

Instructions

Use SYS(6) to determine whether someone pressed CTRL/C while CTRL/C operation was disabled with the RCTRLC function. The SYS(6) values have the meanings shown in the following table.

Value	*Meaning*
0	No CTRL/C combinations were entered while CTRL/C was disabled.
1	At least two CTRL/C characters were entered while CTRL/C was disabled.

Restrictions

The SYS(6) function can take on the value 1 only when CTRL/C is disabled. The CTRLC function sets the value of SYS(6) to 0, possibly losing information. Thus, SYS(6) is only meaningful when RCTRLC is in effect.

The SYS(6) function has no effect in immediate mode.

Errors

None included.

Related

The RCTRLC function disables CTRL/C operation.

The CTRLC function and the end of the program both reenable CTRL/C operation.

References

None.

Examples

None included.

SYS(7,0)

Purpose

The SYS(7,0) function specifies that MINC accepts characters entered from the keyboard in either upper or lower case. This is in fact how MINC normally operates. The SYS(7,0) function exists to reinstate this literal case input if it has been altered with the SYS(7,1) function.

Forms

variable=SYS(7,0)

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric	unknown	required component
=	none	none	required component
SYS(7.0)	none	none	required component

Instructions

Use SYS(7,0) to reinstate MINC's normal method of accepting characters from the keyboard. The value of the variable is unknown after the function has executed. You do not need to use this variable; it is only part of the syntactic form of the function.

Restrictions

The system program that displays READY automatically reinstates mixed case input from the keyboard.

Errors

None included.

Related

The SYS(7,1) function results in conversion of all lowercase characters from the keyboard into the corresponding uppercase characters. (This is equivalent to leaving the CAPS LOCK key pressed in.)

References

None.

Examples

None included.

SYS(7,1)

Purpose

The SYS(7,1) function converts all characters input from the keyboard into their uppercase equivalents.

Forms

variable=SYS(7,1)

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric	variable name	required component
=	none	none	required component
SYS(7.1)	none	none	required component

Instructions

Use the SYS(7,1) function to convert all keyboard input to upper case. This is equivalent to leaving the CAPS LOCK key on the keyboard locked in. Characters appear as uppercase on the screen regardless of which case was entered on the keyboard.

The value of the variable is unknown after the function has executed. You do not need to use this variable; it is only part of the syntactic form of the function.

Using SYS(7,1) can simplify program dialogs by eliminating the need for case conversion of string input.

Restrictions

The SYS(7,1) function operates only in program mode because the system program which displays READY also executes SYS(7,0).

Errors

None included.

Related

The SYS(7,0) function restores the normal mixed case keyboard input for MINC.

References

None.

Examples

None.

TAB

Purpose

The TAB function operates as an argument for a PRINT statement. It generates the number of spaces necessary to move the cursor from its current position to the screen column specified in the TAB argument. The PRINT statement begins printing in the next column.

Forms

TAB(number)

Component	*Component Type*	*Component Value*	*Default Condition*
TAB	none	none	required component
number	numeric expression	0 to 32,767	required component

Instructions

Use TAB to control the appearance of screen displays produced with PRINT statements.

The TAB function is an argument in a PRINT statement. The TAB function causes the cursor to move to the column specified in its argument. Printing resumes with the following column.

If the column number specified is smaller than the current cursor position, the cursor does not move. If the column number is greater than 80, TAB performs repeated subtractions of 80 from the column number until the number is less than or equal to 80. The cursor leaves one blank line for each 80 columns specified.

Follow the TAB function with a semicolon (or leave out the separators) to prevent the PRINT statement from reverting to the normal print zones.

Restrictions

As far as TAB is concerned, there are 80 tab stops in each line, positioned on columns 1 through 80.

The TAB function actually depends on the value of the terminal line length established with the TTYSET system function. If you have set the line width to 132 columns using the TTYSET system function, then the TAB function uses 132 tab stops on each line, positioned in columns 1 through 132. Note that you can set the line width to any width that you choose with the TTYSET system function.

If you use the SETUP mode to change the screen width to 132 columns, the TAB function still thinks there are only 80 columns unless you also use the TTYSET system function to set the width to 132.

Errors

?MINC-F-Arguments in definition do not match function called

Related

The CHR$ function provides some capabilities for formatting screen displays.

The graphic routines provide flexible and powerful methods for formatting screen displays.

The TTYSET system function is the only means in BASIC to set the line width.

References

Book 2, Chapter 3.

Examples

Example PRINT '1'TAB(5)'6'

Result 1 6

Example PRINT '123456789';TAB(4);'0'

Result 1234567890

Example PRINT TAB(0);'1'

Result 1

Example PRINT TAB(0)1

Result 1

Example
```
X=5\Y=20
PRINT TAB(X+Y)X+Y
```

Result 25

TIME

Purpose

When you begin each working session with MINC, MINC prompts you to type in the time of day. You can also use the TIME command whenever MINC displays READY to set the system's clock to the time of day. TIME also causes MINC to display the current system time, but the CLK$ function offers a faster way to check the time.

Forms

TIME hours:minutes:seconds

Component	*Component Type*	*Component Value*	*Default Condition*
TIME	command	none	required component
hours	2 digits	0 to 23	displays current time
minutes	2 digits	0 to 59	0
seconds	2 digits	0 to 59	0

Instructions

MINC's system clock maintains elapsed time extremely accurately. However, because of the amount of time MINC uses to process the TIME command, MINC's internal time of day may differ from the actual time of day by plus or minus 10 seconds. The Examples section describes a specific procedure for reducing this discrepancy as much as possible.

Restrictions

Lab module data transfers in fast mode stop the system clock only during the transfer.

Errors

?KMON-W-Illegal time

The argument values exceed the valid ranges. The current time does not change.

Related

START_TIME
GET_TIME
DATE
DAT$
CLK$
Calendar functions

References

Book 2, Chapter 8.

Examples

The following procedure will set MINC's internal clock to the actual time of day with an accuracy of approximately plus or minus 2 seconds.

Type *TIME xx:xx:07* but don't type RETURN. For "xx:xx" choose a time that is one or two minutes in the future.

When your watch or clock reads *xx:xx:00*, type RETURN.

Since MINC takes about 7 seconds to process the TIME command, the time it establishes on its internal clock will be very close to the actual time of day.

TRM$

Purpose

The TRM$ function trims trailing blanks from its string argument and takes on the value of the trimmed string. The original string does not change.

Forms

TRM$(string)

Component	*Component Type*	*Component Value*	*Default Condition*
TRM$	string	same as argument	required component
string	string expression	any valid value	required component

Instructions

Use TRM$ to remove blanks from the end of any string. This capability is useful when you are concatenating strings to form words or messages without extra embedded blanks.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called at line XX

Related

The functions LEN, SEG$, and POS perform related string processing operations.

References

Book 2, Chapter 8.

Examples

```
Example   A=LEN(TRM$('abcd  ')+'efgh')
          PRINT A

Result      8
```

TTYSET

Purpose

The TTYSET system function specifies the width of a line in BASIC. The TTYSET system function applies to the terminal only.

Forms

variable=TTYSET(255,margin)

Component	*Component Type*	*Component Value*	*Default Condition*
variable	numeric	variable	required component
=	none	none	required component
TTYSET	function name/numeric	unknown	required component
255	numeric literal	255	required component
margin	numeric expression	0 to 255	required component

Instructions

Use TTYSET to define the right margin for BASIC. Normally, BASIC uses an 80 column line for listing programs and displaying output from PRINT statements, even if you have used the SETUP mode to lengthen the lines to 132 columns. If you use the TTYSET system function, BASIC uses the value of the margin argument as the right margin.

The column specified as the margin is not itself part of the line. That is, a line with right margin 81 can contain at most 80 characters.

If the margin argument is 0, then the margin does not change from its previous setting.

The standard margin setting for MINC is currently 80.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called

The margin argument in greater than 255.

Related

The TAB function uses the value set by the TTYSET system function.

References

None.

Examples

None included.

TYPE

Purpose

To display any file you have typed or any ASCII files your programs have created, use the TYPE command when MINC displays READY.

Forms

TYPE filespec

No spaces can appear after the file specification.

Component	*Component Type*	*Component Value*	*Default Condition*
TYPE	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name required .typ .BAS

Instructions

Complete the command with a single file specification. MINC displays the file line by line. You cannot display any part of the file again after MINC scrolls it off the top of your screen, but you can interrupt scrolling by pressing the NO SCROLL key. MINC waits until you press the NO SCROLL key again to continue the display.

Restrictions

MINC does not stop you from using the TYPE command to display non-ASCII files, such as compiled files (.BAC) and virtual array files. However, the information is not in a form that you can read. It looks very confusing on the screen.

Errors

?UTILITY-F-File not found

The file named in the TYPE command does not exist on the volume.

Related

COPY
LIST

References

Book 2, Chapter 4.

Examples

None included.

UNSAVE

Purpose

Whenever MINC displays READY, use the UNSAVE command to erase a file from a diskette. The equivalent statement is the KILL statement. When MINC executes an UNSAVE command, it erases the directory entry for the file you have specified. The space on your diskette that actually holds the file is immediately available for another file with the same name or a different name.

Note that the KILL statement defaults to .DAT files where the UNSAVE command defaults to .BAS files. Thus, the KILL statement is primarily used within programs to delete temporary data files where the UNSAVE command is primarily used to delete BASIC programs.

Forms

UNSAVE filespec

Component	*Component Type*	*Component Value*	*Default Condition*
UNSAVE	command	none	required component
filespec	characters	dev:name.typ	dev: SY0: name required .typ .BAS

Instructions

Complete the UNSAVE command with a file name. The file types .SYS, .SAV, .COM, and .BAD are protected, and MINC displays the following message if you try to erase a file with one of those file types.

```
?MINC-F-Use another file type; SYS, SAV, COM and BAD are protected
```

After MINC completes an UNSAVE command, you cannot recover the file you have erased.

Restrictions

Be careful specifying the file name because the .typ defaults to .BAS. You might actually delete your BASIC program when trying to delete the compiled version.

Errors

```
?MINC-F-Specified or default volume does not have file named
```

The file that you tried to UNSAVE does not exist on the volume.

UNSAVE

Related

KILL
COLLECT
DIR

References

Book 2, Chapter 4.

Examples

See Book 2.

VAL

Purpose

The VAL function converts a string representation of a number to its numeric equivalent. The VAL function takes on the converted numeric value.

Forms

VAL(string)

Component	*Component Type*	*Component Value*	*Default Condition*
VAL	numeric	any valid value	required component
string	string expression	string representation of a number	required component

Instructions

Use VAL to convert a character string to the numeric value it represents. This is useful when you need to input both letters and digits to string variables but want to convert the digits to their numeric values.

The character string can contain the digits 0 through 9, a minus or plus sign, a decimal point, and the power of 10 indicator, E.

Restrictions

None included.

Errors

?MINC-F-Arguments in definition do not match function called

The string argument contains a character or format that would produce an invalid numeric value.

Related

**STR$** The STR$ function converts a numeric value to the corresponding string representation. STR$ and VAL reverse each other's effects, as shown in the following identity:

string = STR$(VAL(string))

**ASC, CHR$** The ASC and CHR$ conversion functions are not related to VAL. ASC converts a single character to its numeric ASCII code. CHR$ converts a numeric ASCII code to a single character string.

References

Book 2, Chapter 8.

Examples

Example A=VAL('12345')
PRINT A

Result 12345

Example PRINT VAL('1234567')

Result 1.23457E+06

Example PRINT VAL('12e6')

Result 1.20000E+07

Example PRINT VAL('10E1')

Result 100

Example PRINT VAL('10.0 E+01')

Result 100

VERIFY

Purpose

Mass storage volumes are durable, but not indestructible. Use the VERIFY command to obtain a report of the number of bad blocks on a volume and the names of any files, if any, that are stored across bad blocks.

You may never find any bad blocks if you are reasonably careful about storing and handling your storage volumes. If MINC encounters a bad block in any file while executing a DUPLICATE command you can use the VERIFY command to find out which block is bad. In some cases, you might be able to use the editor to reconstruct the data in the bad block after you have copied the file to a new location. See Error Recovery.

Forms

VERIFY dev:

Component	*Component Type*	*Component Value*	*Default Condition*
VERIFY	command	none	required component
dev:	characters	SY0: SY1:	SY0:

Instructions

Complete the command with a device abbreviation. The default device is SY0:.

If a bad block develops in one of the MINC system files, the VERIFY report lists the file name. You can tell that this file is a MINC system file because MINC system files have one of the protected file types: .BAD, .SAV, .SYS, or .COM.

To recover from a bad block in a system file, see the section entitled Error Recovery.

Restrictions

The VERIFY report does not tell you which device was verified.

If there were no bad blocks, the VERIFY command produces the following report:

There were no bad blocks found

Errors

?UTILITY-F-Illegal command

You tried to verify an invalid device, such as SY2:, or a nonexistent device, such as LP:.

?UTILITY-F-Error reading directory

You forgot to close the door to the device.

There are bad blocks in the directory of the volume.

?MINC-F-Specify only one file name

You can only verify one volume at a time.

?MINC-F-Specify device only; no file name or file type is allowed

Related

DUPLICATE
INITIALIZE
Error Recovery

References

Book 2, Chapter 4.

Examples

In the following example, one bad block was marked in the FILE.BAD file during the initialization. The other bad block is in the user's program file named SINES.BAS.

```
READY

VERIFY

Bad Blocks      Type        Filename        Rel Blk
414             Hard        FILE.BAD        0
417             Hard        SINES.BAS       0
```

To recover the diskette, the user must perform the following steps.

1. Duplicate the diskette using the DUP command.

2. Fix the new copy of SINES.BAS using the keypad editor. (See the Error Recovery section.)

3. Initialize the bad diskette with the INI command to mark the bad block in the FILE.BAD file. Then the user can use the bad diskette again (it is no longer bad).

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____ Telephone_____

City_____ State_____ Zip Code_____
or
Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MR1-2/E37
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line