RC BASIC
Programming Guide

3600 / 7000

RC BASIC

Reserved Words in the RC BASIC Language

The use of the words is denoted by the following abbreviations: AO (arithmetic operator), C (command), F (function), LO (logical operator), NI (not implemented), and S (statement).

The section (chapter) number indicates where the sole, first, or principal explanation of the word will be found.

Word	Use	Section	Word	Use	Section	Word	Use	Section
ABS	F	4.2	FALSE	NI		RANDOMIZE	S,C	3.27
AND	LO	2	FILE	S,C	8	READ	S,C	3.28
ATN	F	4.3	FNA - FNÅ	F	3.5	RELEASE	C,S	7.7
AUTO	C	9.3	FOR	S	3.11	REM	S	3.29
						RENAME	S,C	8.15
BATCH	C	9.4	GOSUB	S	3.12	RENUMBER	C	9.14
BYE	C,S	9.5	GOTO	S	3.13	REPEAT	S	3.30
						RESTORE	S,C	3.31
CALL	NI					RETURN	S	3.12
CASE	S	3.2	IDN	S,C	6.8	RND	F	4.9
CHAIN	S,C	3.3	IF	S,C	3.14	RUN	C	9.15
CHR	F	5.2	INIT	C	7.4	RUNL	C	9.15
CLOSE	S,C	8.2	INPUT	S,C	3.17			
CON	S,C	6.7	INT	F	4.7			
CON	C	9.6	INV	S,C	6.10	SAVE	C,S	9.16
CONL	C	9.6				SCRATCH	C	B.5
CONNECT	C,S	7.2				SGN	F	4.10
COPY	C	7.3	LEN	F	5.3	SIN	F	4.11
COS	F	4.4	LET	S,C	3.18	SIZE	C	9.18
CREATE	S,C	8.3	LIST	C	9.9	SQR	F	4.12
			LOAD	C	9.10	STEP	S	3.11
DATA	S	3.4	LOCK	C	7.5	STOP	S	3.33
DEF	S	3.5	LOG	F	4.8	SYS	F	4.13
DELAY	S	3.6	LOOKUP	C	7.6			
DELETE	S,C	8.4						
DET	F	6.6				TAB	C,S	9.19
DIM	S,C	3.7	MAT	S,C	6	TAN	F	4.14
DIV	AO	2	MOD	AO	2	THEN	S,C	3.14
DO	S	3.36				TIME	C	B.6
			NEW	C,S	9.11	TINPUT	NI	
ELSE	S	3.16	NEXT	S	3.11	TO	S	3.11
END	S	3.8	NOT	LO	2	TRN	S,C	6.13
ENDCASE	S	3.2				TRUE	NI	
ENDIF	S	3.15	OF	S	3.2			
ENDPROC	S	3.26	ON	S	3.22	UNTIL	S	3.30
ENDWHILE	S	3.36	OPEN	S,C	8.11	USERS	C	7.8
ENTER	C,S	9.7	OR	LO	2	USING	S	3.25
EOF	F	8.5	ORD	F	5.4			
EOJ	C	B.4				WHEN	S	3.2
ERR	S	3.20	PAGE	C,S	9.12	WHILE	S	3.36
ESC	S	3.21	PRINT	S,C	3.24	WRITE	S,C	8.16
EXEC	S	3.10	PROC	S	3.26			
EXP	F	4.5	PUNCH	C	9.13	ZER	S,C	6.14

RC BASIC

A Structured Educational Language (COMAL)

PROGRAMMING GUIDE

A/S REGNECENTRALEN
Development Division
Documentation Department

First Edition
September 1977
RCSL 42-i 0671

Authors: Stig Møllgaard and Pierce Hazelton

Keywords: RC 3600, RC 7000, MUS, DOMUS, RC BASIC, Programming Guide

Abstract: This guide describes the RC BASIC language implemented for RC 3600 and RC 7000 minicomputers.

Supporting documents: RC BASIC System Logical Disc Formatting Program Operating Guide (RCSL 43-G 45069).

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without prior notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and shall not be responsible for any damages caused by reliance on any of the materials presented.

Printed by A/S Regnecentralen, Copenhagen

Foreword

RC BASIC is a structured educational language, implemented by A/S Regnecentralen to run on RC 3600 and RC 7000 minicomputers.

The education sector has long felt the need for a suitable programming language: one which is simple and comprehensible, yet sufficiently advanced to permit demonstration of important programming principles.

Until now the BASIC language has generally been used, for despite its deficiencies as regards advanced programming, BASIC, precisely because of its comprehensibility and convenient conversational form, has fulfilled a basic requirement in educational applications.

In recent years proposals have been put forth for better educational languages, among them COMAL (Common Algorithmic Language).

COMAL possesses all the features that have made BASIC popular; in fact, COMAL includes almost all of the facilities found in existing versions of BASIC, and users can therefore run almost any program written in BASIC on a computer that runs COMAL.

COMAL, however, contains significant extensions to the BASIC language.

COMAL was proposed primarily to accommodate users desiring more advanced control facilities than those found in BASIC. Thus the designers of COMAL, inspired by the PASCAL language, have introduced five new control structures: REPEAT-UNTIL, WHILE-DO-ENDWHILE, IF-THEN-(ELSE)-ENDIF, and CASE-OF-WHEN-ENDCASE.

COMAL also contains several other important extensions, such as the possibility of using long variable names (as many as eight characters), all of which contribute to making COMAL programs clearer and more readable than programs written in BASIC.

The COMAL language, which was designed by Børge Christensen, Government Teachers' College, Tønder, Denmark, in collaboration with Benedict Løfstedt, Århus University, is incorporated in RC BASIC.

Contents

APPENDICES

1 Introduction

1.1 General information

The RC BASIC programming language provides facilities for:

Writing structured programs.
Executing programs in interactive mode.
Running jobs in batch mode.
Performing file input/output.
Performing matrix operations.
Manipulating strings.
Formatting output.
Performing desk calculator functions.

RC BASIC runs under the RC operating system MUS (Multiprogramming Utility System) or DOMUS (Disc Operating Multiprogramming Utility System).

This programming guide describes the syntax and semantics of RC BASIC statements, commands, and functions.

Those who have access to terminals can use RC BASIC in interactive mode, but it is also possible to execute programs written on mark-sense cards in batch mode (see App. B).

1.2 RC BASIC programs

An RC BASIC program consists of a number of statements. Each statement begins with a line number, in the range 1 to 9999, which determines the order in which the statement will be executed. The rest of the statement is made up of one or more RC BASIC words (see below), with or without arguments. Each statement is written on a separate line.

The user terminates each statement line by pressing the RETURN key. This generates an automatic line feed in addition to the carriage return. (Note: The "carriage return" separator referred to in this guide is the RETURN key, not the ASCII character Carriage Return). If the user discovers a typing error before he has pressed the RETURN key, he can delete the last character typed by pressing the RUBOUT key (repeatedly, if need be) or delete the entire line by pressing the ESCape key.

Some RC BASIC words, such as END, STOP, or CLOSE, can be used alone to perform an operation; others require one or more arguments, on which the operations are performed. Thus the word READ, for example, cannot be used alone; READ must have at least one argument, viz. the name of a variable to which a value is to be assigned (e.g. READ PRICE).

The user may enter program statements in any order. The system will arrange them by ascending line numbers.

When a program is run, the statements are executed one by one in ascending line number order, usually beginning with the lowest numbered statement. The sequential execution of statements may be interrupted by a "control transfer statement," such as ENDWHILE, EXEC, ENDPROC, or GOTO.

Many of the RC BASIC statements, which are described in Chapters 3, 6, and 8, may also be used as keyboard commands (see App. C). When a statement is used as a command, it is entered without a preceding line number and terminated by pressing the RETURN key, whereupon the system executes it immediately.

Still other RC BASIC words can only be used alone, i.e. they cannot be part of a statement, but are used solely as commands. LIST and RUN are examples of such words.

When a program has been entered, it will remain in core memory until the user clears it by means of a NEW command.

New statement lines can be inserted anywhere in a program. Existing statements can be deleted, by typing the statement line number and pressing the RETURN key, or corrected, simply by entering a new statement with the same line number.

The currently loaded program can be executed by means of a RUN/RUNL command.

1.3 ESCape key

Pressing the ESCape key during program execution will cause interruption of the program, unless an ON-ESC statement has been executed (see Ch. 3). Control will be returned to interactive mode, and the system will output the following on the user's terminal:

```
STOP
AT <xxxx>
*
```

where <xxxx> is the line number of the statement at which the program was interrupted. The asterisk (*) prompt indicates that the user may enter a command or a program statement. Program execution can be resumed by means of a CON/CONL or RUN/RUNL <line no.> command.

Pressing the ESCape key on an idle terminal (e.g. after system start-up or a BYE command) will place the terminal in interactive mode.

When a terminal is in batch mode, the ESCape key has a special function (see App. B).

1.4 Descriptions of statements, commands, and functions

Descriptions of the statements, commands, and functions in the RC BASIC language will be found in Chapters 3 through 9. These descriptions have the following form:

```
x.y  RC BASIC WORD
     Format
     Use
     Remarks
     Example
```

where

x.y : Chapter.Section

RC BASIC WORD : One or more reserved RC BASIC words

Format : The generalized format (syntax) of the statement, command, or function.

This format, which is explained in detail below, must be used when the statement, command, or function is entered from the terminal, otherwise an error message (usually 0002: SYNTAX ERROR) will result.

Use : Indicates whether the RC BASIC word is used as a statement, command, or function, and describes the operation or operations which it performs.

Remarks : Contains remarks concerning the use of the statement, command, or function, including rules, precautions, program operation, and the like.

Example : The use of most of the statements, commands, and functions is illustrated by one or more examples, which usually consist of small programs, followed by the output produced when the program was executed.

As many of the programs were listed on the line printer, and the resulting output was directed to the line printer, the commands used (viz. LIST "$LPT" and RUNL) do not appear in these examples.

In a few examples, for clarity's sake, the text entered by the user is underlined and followed by the symbol) to denote that the user has terminated the line by pressing the RETURN key, e.g. ? 5,6,7,8)

1.5 Formats used in descriptions

Capital letters in the generalized format denote literal entries.

Any parentheses should be inserted as indicated.

Braces ({}) indicate a choice of the items enclosed.

Brackets ([]) indicate that the enclosed items are optional.

An ellipsis (...) indicates that the preceding argument may be repeated.

Several abbreviations are used in the formats to represent common terms. All abbreviations in a format are explained immediately beneath it, while the terms represented are defined in the appropriate chapters of this guide. The most frequently occurring abbreviations are:

<var> : The name of a numeric variable, with or without subscripts.

<svar> : The name of a string variable, with or without subscripts.

<expr> : A numeric, relational (Boolean), or string expression (see Ch. 2).

<slit> : A string literal (string constant), i.e. a sequence of characters enclosed within quotation marks (").

<val> : A numeric constant.

<line no.> : A statement line number in the range 1 to 9999.

<statements> : One or more RC BASIC statements.

<mvar> : The name of a matrix variable.

<ldname> : The name of a logical disc.

<filename> : The name of a disc file or a device.

<device> : The name of a device.

<file> : A numeric expression which evaluates to a number in the range 0 to 7 (the number of a user file).

As an example, consider the generalized format of the PRINT statement (see Ch. 3):

```
          [ {<expr>} [       {<expr>} ]     ]
{ ;     } [ {<slit>} [ { , } {<slit>} ]     ] [ { , } ]
{ PRINT } [ {<svar>} [ { ; } {<svar>} ] ... ] [ { ; } ]
1       1 2 5      5 6 7   7          6     2 3 4   4 3
```

The PRINT statement begins with the word PRINT. As PRINT is frequently used, one can write a semicolon (;) instead of the word PRINT. This is indicated by the pair of braces 1-1.

The pair of brackets 2-2 indicates that the PRINT statement need not have an argument. PRINT may optionally (brackets 3-3) be followed by a comma or a semicolon (braces 4-4).

An argument may be of the type <expr>, <slit>, or <svar>, as indicated by the pair of braces 5-5. If there is more than one argument (brackets 6-6 and the ellipsis), the arguments should be separated by a comma or a semicolon (braces 7-7).

2 RC BASIC Arithmetic

2.1 Numbers

An RC BASIC number may be in the range:

$$5.4 * 10^{-79} < n < 7.2 * 10^{75}$$

Numbers may be expressed as integers, as floating-point numbers, or in exponential form (E-type notation).

In the conversion of numeric data, e.g. by a PRINT statement (see Ch. 3), any floating-point or integer number that contains six digits or less is formatted without using exponential form. A floating-point or integer number that requires more than six digits is printed in the following E-type notation:

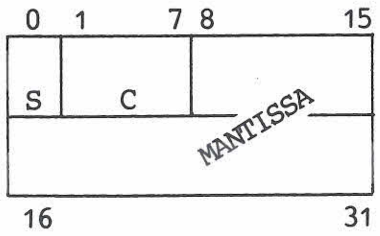
```
<sign>n.nnnnnE<sign>XX
```

where n.nnnnn is an unsigned number carried to five decimal places with trailing zeroes suppressed, E means "times 10 to the power of," and XX represents an unsigned exponential value.

Number	Output format
2,000,000	2E+06
108.999	108.999
.0000256789	2.56789E-05
24E10	2.4E+11

2.2 Internal representation of numbers

Internally, floating-point numbers are stored in two consecutive 16-bit words having the form:



where: S is the sign of the mantissa (0 = positive, 1 = negative); the mantissa is a normalized six-digit hexadecimal fraction; and C is the characteristic and an integer expressed in excess 64 code.

2.3 Variables

A numeric variable name (shown in the statement descriptions as <var>) consists of a single letter followed by from 0 to 7 digits or letters, for example:

Legal names	Illegal names
I	$PRICE
INTEREST	6I
AMOUNT	MEANVALUE
PRICE	

In addition to numeric variables, string variables (shown in the statement descriptions as <svar>) are also permitted in RC BASIC (see Ch. 5).

2.4 Arrays

An array represents an ordered set of values. Each member of the set is called an array element. An array can have either one or two dimensions. An array name consists of a single letter followed by from 0 to 7 digits or letters.

2.4.1 Array elements

Each of the elements of an array is identified by the name of the array followed by a parenthesized subscript, for example:

```
ITEMNO(1), ITEMNO(2), ..., ITEMNO(8), ITEMNO(9)
```

For a two-dimensional array, the first number gives the number of the row and the second gives the number of the column for each element. Thus the elements of the array C(2,3) would be:

```
C(1,1)      C(1,2)      C(1,3)
C(2,1)      C(2,2)      C(2,3)
```

2.4.2 Declaring an array

An array must be declared in a DIM statement (see Ch. 3), which gives the name of the array and its dimensions.

The lower bound of a dimension is always 1. The upper bound is given in the DIM statement, two upper bounds being separated by a comma (,). Dimensional information is enclosed in parantheses immediately following the name of the array in the DIM statement, for example:

```
5 DIM SET(15), AMOUNT(2,3)
```

The total number of elements in an array may not exceed 32 767. Available memory will normally impose a limit, however.

2.5 Expressions

An expression (shown in the statement descriptions as <expr>) may be composed of parentheses, constants and variables (numeric or string), and functions, linked together by operators.

2.5.1 Numeric expressions

A numeric expression may be composed of numeric variables and constants and (numeric) functions, linked together by arithmetic operators.

2.5.2 Arithmetic operators

The arithmetic operators are as follows:

+	: monadic + (A+(+B))
-	: monadic - (A+(-B))
↑	: exponentiation (A↑B)
*	: multiplication (A*B)
/	: division (A/B)
MOD	: modulus calculation (A MOD B)
DIV	: integer division (A DIV B)
+	: addition (A+B)
-	: subtraction (A-B)

The operators +, -, *, /, and ↑ are familiar to most, but DIV and MOD may require some explanation.

2.5.2.1 DIV operator. The result of an integer division A DIV B is equal to:

```
SGN(A/B)*INT(INT(ABS(A))/INT(ABS(B)))
```

The SGN(X), INT(X), and ABS(X) functions are described in Chapter 4.

Examples

```
 11 DIV 4 = +1 x 2 = 2
-11 DIV 4 = -1 x 2 = -2
```

2.5.2.2 MOD operator. The result of a modulus calculation A MOD B is equal to:

```
SGN(A)*INT(ABS(A))-A DIV B*SGN(B)*INT(ABS(B))
```

The SGN(X), INT(X), and ABS(X) functions are described in Chapter 4.

Examples

```
 11 MOD 4 = 11 - 2 x 4 = 3
-11 MOD 4 = -11 - (-2) x 4 = -3
```

2.5.3 Priorities of arithmetic operators during program execution

As a general rule, numeric expressions are evaluated from left to right.

The arithmetic operators, however, have different priorities, for which reason the following exceptions to this rule apply:

1. Numeric expressions enclosed within parentheses are always evaluated before non-parenthesized expressions. If expressions are nested, the innermost expression is evaluated first.

2. Functions are evaluated next.

3. The priorities of the arithmetic operators are as follows:

First:	monadic plus and monadic minus
Second:	exponentiation
Third:	multiplication, division, modulus calculation, and integer division

Fourth: addition and subtraction

4. When two operators have the same priority, evaluation proceeds from left to right.

The following two examples should help clarify the principles according to which numeric expressions are evaluated.

$1 + 2 - 3 \text{ x } \underbrace{4 \uparrow 2} \text{ DIV } 5 + 6$

1: $1 + 2 - \underbrace{3 \text{ x } 16} \text{ DIV } 5 + 6$

2: $1 + 2 - \underbrace{48 \text{ DIV } 5} + 6$

3: $\underbrace{1 + 2} - 9 + 6$

4: $\underbrace{3 - 9} + 6$

5: $\underbrace{-6 + 6}$

6: 0

$(1 + \underbrace{(2 - 3)} \text{ x } 4 \uparrow 2) \text{ DIV } 5 + 6$

1: $(1 \underbrace{+ (-1)} \text{ x } 4 \uparrow 2) \text{ DIV } 5 + 6$

2: $(1 -1 \text{ x } \underbrace{4 \uparrow 2}) \text{ DIV } 5 + 6$

3: $(1 \underbrace{-1 \text{ x } 16}) \text{ DIV } 5 + 6$

4: $\underbrace{(1 - 16)} \text{ DIV } 5 + 6$

5: $\underbrace{-15 \text{ DIV } 5} + 6$

6: $\underbrace{-3 + 6}$

7: 3

2.5.4 AND, OR, and NOT operators

The operators AND, OR, and NOT are analogous to *, +, and -; but whereas *, +, and - have numeric arguments, AND, OR, and NOT have Boolean arguments. A Boolean argument can have two values: true, corresponding to 1 (or <> 0), and false, corresponding to 0.

NOT operates on one argument only, and means logical negation, for example:

A	NOT A
TRUE	FALSE
FALSE	TRUE

AND operates on two arguments, and means logical multiplication (logical and), for example:

A	B	A AND B
FALSE (0)	FALSE (0)	FALSE (0 x 0 = 0)
FALSE (0)	TRUE (1)	FALSE (0 x 1 = 0)
TRUE (1)	FALSE (0)	FALSE (1 x 0 = 0)
TRUE (1)	TRUE (1)	TRUE (1 x 1 = 1)

OR operates on two arguments, and means logical addition (logical or), for example:

A	B	A OR B
FALSE (0)	FALSE (0)	FALSE (0 + 0 = 0)
FALSE (0)	TRUE (1)	TRUE (0 + 1 = 1)
TRUE (1)	FALSE (0)	TRUE (1 + 0 = 1)
TRUE (1)	TRUE (1)	TRUE (1 + 1 <> 0)

The priorities of these three operators are:

First:	NOT
Second:	AND
Third:	OR

2.5.5 Relational expressions

A relational expression is composed of two expressions of the same type, i.e. both numeric or both string, linked together by a relational operator.

2.5.6 Relational operators

The relational operators are as follows:

<	: less than
<=	: less than or equal to
=	: equal to
>=	: greater than or equal to
>	: greater than
<>	: not equal to

As mentioned above, all relational operators have two arguments. The arguments are compared, and the result of the comparison is always either true or false. Thus the notation

$0 < X < 10$

is illegal, as this would result in a comparison of

0 and X (true or false)

followed by a comparison of

(true or false) and 10

which is meaningless. The correct notation is:

```
0 < X  AND  X < 10
```

Examples

1 < 10	is true.
1 > 10	is false.
"JOHN" < "PETER"	is true (see Ch. 5).
1 < 10 AND 20 < 30	is true (1 x 1 = 1).
10 < 5 OR 2 < 11	is true (0 + 1 = 1).
NOT 1 < 4	is false.

2.5.7 Priorities of arithmetic, Boolean, and relational operators

In compound expressions, the priorities of arithmetic, Boolean, and relational operators are as follows:

First: monadic plus and monadic minus
Second: exponentiation
Third: multiplication, division, modulus calculation, and integer division
Fourth: addition and subtraction
Fifth: relational operators (<>, <, <=, =, >=, >)
Sixth: NOT
Seventh: AND
Eighth: OR

2.5.8 String expressions

A string expression (see further Ch. 5) may be any of the following:

1. A string variable, e.g. ANSWER$
2. A string literal, e.g. "PETER"
3. The CHR(X) function, e.g. CHR(65)
4. A concatenation of the above items, e.g. "JOHN SMITH",ADDRESS$

3 RC BASIC Statements

3.1 BYE

For description, see Chapter 9.

3.2 CASE-WHEN-ENDCASE

Format

```
CASE <expr> OF
  [<statements-0>]
WHEN <expr> [,<expr>] ...
   <statements-1>
┌                          ┐
│     .                    │
│     .                    │
│     .                    │
│WHEN <expr> [,<expr>] ... │
└   <statements-n>         ┘
ENDCASE [<comment>]
```

```
             <expr>:  an expression.
     <statements-0>:  a block of statements.
       .
       .
       .
     <statements-n>:  a block of statements.
          <comment>:  a text comment.
```

Use

As a statement to execute one of several block of statements depending on the value of an expression.

Remarks

1. Rules

   a. <expr> may be an expression of any kind.

   b. For every CASE statement there must be at least one corresponding WHEN statement and one ENDCASE statement.

c. If a block of statements belonging to a CASE construction is entered from outside the construction, the error message 0062: WHEN WITHOUT CASE or 0061: ENDCASE WITHOUT CASE will be output when WHEN or ENDCASE is encountered.

2. Program operation

a. The expression in the CASE <expr> OF statement is evaluated.

b. The expressions in the WHEN <expr> [,<expr>] statements are evaluated one by one until a value is found which is equal to the value obtained in step a. If this value is found in the ith WHEN statement, <statements-i> is executed.

c. Execution continues until a WHEN or ENDCASE statement is encountered; after this, control is transferred to the first statement following the ENDCASE statement.

d. If a matching value is not found in step b, <statements-0> is executed. If <statements-0> is not present, the error message 0059: CASE WITHOUT WHEN, CASE ERROR will be output.

3. Nested constructions

CASE-WHEN-ENDCASE constructions may be nested to any depth.

4. The word ENDCASE may be followed by a comment.

Example 1

```
0010 FOR I=1 TO 5
0020   CASE I OF
0030     PRINT "CASE ERROR - I"
0040   WHEN 1,3+1,6
0050     FOR J=3 TO 5
0060       CASE J OF
0070         PRINT "CASE ERROR - J"
0080       WHEN 3
0090         PRINT "I,J =";I;J
0100       WHEN 4
0110         PRINT "J,I =";J;I
0120       ENDCASE
0130     NEXT J
0140   WHEN 2
0150     PRINT "I =";I
0160   WHEN 3
0170     PRINT "I =";I
0180   ENDCASE
0190 NEXT I
0200 STOP
```

```
I,J = 1  3
J,I = 4  1
CASE ERROR - J
I = 2
I = 3
I,J = 4  3
J,I = 4  4
CASE ERROR - J
CASE ERROR - I
```

Comment (1)

Shows a nested CASE construction.

Example 2

```
0010 DIM MONTH$(10)
0020 FOR MONTHNR=1 TO 13
0030   CASE MONTHNR OF
0040     PRINT "ILLEGAL NUMBER: ";MONTHNR
0050     GOTO 0330
0060   WHEN 1
0070     LET MONTH$="JANUARY"; DAYS=31
0080   WHEN 2
0090     LET MONTH$="FEBRUARY"; DAYS=28
0100   WHEN 3
0110     LET MONTH$="MARCH"; DAYS=31
0120   WHEN 4
0130     LET MONTH$="APRIL"; DAYS=30
0140   WHEN 5
0150     LET MONTH$="MAY"; DAYS=31
0160   WHEN 6
0170     LET MONTH$="JUNE"; DAYS=30
0180   WHEN 7
0190     LET MONTH$="JULY"; DAYS=31
0200   WHEN 8
0210     LET MONTH$="AUGUST"; DAYS=31
0220   WHEN 9
0230     LET MONTH$="SEPTEMBER"; DAYS=30
0240   WHEN 10
0250     LET MONTH$="OCTOBER"; DAYS 31
0260   WHEN 11
0270     LET MONTH$="NOVEMBER"; DAYS 30
0280   WHEN 12
0290     LET MONTH$="DECEMBER"; DAYS 31
0300   ENDCASE
0310   PRINT TAB(10-LEN(MONTH$));MONTH$;
0320   PRINT " HAS";DAYS;"DAYS."
0330 NEXT MONTHNR
0340 STOP
```

```
  JANUARY HAS 31 DAYS.
 FEBRUARY HAS 28 DAYS.
    MARCH HAS 31 DAYS.
    APRIL HAS 30 DAYS.
      MAY HAS 31 DAYS.
     JUNE HAS 30 DAYS.
     JULY HAS 31 DAYS.
   AUGUST HAS 31 DAYS.
SEPTEMBER HAS 30 DAYS.
  OCTOBER HAS 31 DAYS.
 NOVEMBER HAS 30 DAYS.
 DECEMBER HAS 31 DAYS.
ILLEGAL NUMBER:  13
```

Example 3

```
0010 DIM ANSWER$(20)
0020 PROC GETANSWR
0030   REM THE PROCEDURE ACCEPTS ONE OF THREE POSSIBLE
0040   REM ANSWERS: YES, NO, OR DON'T KNOW
0050   REPEAT
0060     LET ERROR=0
0070     INPUT ANSWER$
0080     PRINT ANSWER$
0090     CASE ANSWER$ OF
0100       PRINT "ERROR, RETYPE"
0110       LET ERROR=1
0120     WHEN "YES"
0130       LET YES=YES+1
0140     WHEN "NO"
0150       LET NO=NO+1
0160     WHEN "DON'T KNOW","DO NOT KNOW"
0170       LET DONTKNOW=DONTKNOW+1
0180     WHEN "END"
0190       LET FINIS=1
0200     ENDCASE
0210   UNTIL NOT ERROR
0220 ENDPROC
0230 REM
0240 REM MAIN PROGRAM
0250 REM PRINT QUESTION, CHECK ANSWER
0260 REM BY MEANS OF THE PROCEDURE
0270 REM GETANSWR
0280 REM
0290 LET FINIS=0; YES=0; NO=0; DONTKNOW=0
0300 REPEAT
0310   PRINT "QUESTION"
0320   EXEC GETANSWR
0330 UNTIL FINIS
0340 PRINT "<13><10><10>"," YES"," NO","DON'T KNOW"
0350 PRINT ,YES,NO,DONTKNOW
0360 STOP
```

```
QUESTION
NO
QUESTION
YES
QUESTION
YES
QUESTION
DO NOT KNOW
QUESTION
NO
QUESTION
DON'T KNOW
QUESTION
NOT
ERROR, RETYPE
NO
QUESTION
YESE
ERROR, RETYPE
YES
QUESTION
END
                  YES            NO          DON'T KNOW
                   3              3              2
```

Comment (3)

For PROC-ENDPROC, see Section 3.26.

3.3 CHAIN

Format

```
CHAIN <filename> [THEN GOTO <lineno.>]
```

<filename>: a disc file or a device expressed as a string literal or by means of a variable.

<line no.>: the line number in the program referred to by <filename> from which execution is to begin.

Use

As a statement or command to run the SAVEd program referred to by <filename> when the CHAIN statement is encountered in the user's program.

Remarks

1. When a CHAIN statement is encountered in a program, it stops execution of that program, loads a previously SAVEd program (see SAVE, Ch. 9) from the disc file or the device specified by <filename>, and begins execution of the SAVEd program.

2. If the SAVEd program is on disc, the system searches the logical disc to which the terminal is connected for <filename> (see Ch. 8). If <filename> is not found, the system outputs the error message 0100: FILE UNKNOWN.

3. If <filename> is found, the user's currently running program is cleared from core memory and the SAVEd program is loaded into core memory from <filename>. If <filename> is not found, the current program remains in core memory.

4. The newly loaded program is run from its lowest numbered statement, unless the THEN GOTO <line no.> argument is given in the CHAIN statement to specify another line number from which execution is to begin.

5. The CHAIN statement is typically used to divide a large program into smaller programs or to run independent programs from a main program on the basis of conditional statements.

6. CHAIN may also be used as a command, in which case it has the same effect as LOAD (see Ch. 9), i.e. the SAVEd program is loaded, but not executed.

Example 1

```
0010 INPUT "SELECT PROGRAM NUMBER : ",NUMBER
0020 CASE NUMBER OF
0030   PRINT "ILLEGAL NUMBER"
0040 WHEN 1
0050   CHAIN "PROGRAM1"
0060 WHEN 2
0070   CHAIN "PROGRAM2"
0080 WHEN 3
0090   CHAIN "PROGRAM3"
0100 ENDCASE
```

Comment (1)

The user selects a program by typing a number.

Example 2

```
0010 DIM NAME$(10)
0020 INPUT "SELECT PROGRAM : ",NAME$
0030 CHAIN NAME$
0040 END
```

Comment (2)

The user selects a program by typing its name, i.e. the name of the disc file or the device from which the program is to be CHAINed.

3.4 DATA

Format

```
     ⎧<val> ⎫ ⎡⎧,<val> ⎫⎤
DATA ⎩<slit>⎭ ⎣⎩,<slit>⎭⎦ ...
```

<val>: a numeric value.
<slit>: a string literal.

Use

As a statement to provide values to be read into variables appearing in READ statements.

Remarks

1. The DATA statement is non-executable.

2. The values appearing in a DATA statement or statements form a single list. The first element in this list is the first item in the lowest numbered DATA statement. The last element in the list is the last item in the highest numbered DATA statement.

3. Both numbers and string literals may appear in a DATA statement. Each value in a DATA statement list must be separated from the next value by a comma.

Example

```
100 DATA 1,17,"AB,CD",-1.3E-13
```

See further READ (Sect. 3.28).

3.5 DEF

Format

```
DEF FN<a>(<d>) = <expr>
```

<a>: a letter.

<d>: a dummy numeric variable, which may appear in <expr>.

<expr>: a numeric expression, which may contain the variable <d>.

Use

As a statement to permit the user to define as many as 29 different functions, which can be referenced repeatedly throughout a program. Each function returns a numeric value.

Remarks

1. The name of the defined function must be the two letters FN followed by a single letter, viz. A, B, ..., Z, Æ, Ø, or Å.

2. The dummy variable named in the DEF statement is not related to any variable in the program having the same name; the DEF statement simply defines the function and does not cause any calculation to be carried out.

3. In the function definition, <expr> may be any legal numeric expression and may include other user-defined functions. Functions may be nested to a depth of seven.

4. Function definition is limited to a single-line DEF statement. Complex functions that require more that one program statement should be constructed as subroutines or procedures.

Example 1

```
* LIST
0010 DEF FNP(X)=X↑2+2*X+2
0020 FOR I=1 TO 5
0030   PRINT FNP(I);
0040 NEXT I

* RUN
 5  10  17  26  37
END
AT 0040
*
```

Comment (1)

Calculates
P(X) = X↑2+2*X+2 for
different values of X.

Example 2

```
* LIST
0010 DEF FND(X)=X*SYS(14)/180
0020 DEF FNS(X)=SIN(FND(X))
0030 DEF FNC(X)=COS(FND(X))
0040 FOR DEGR=0 TO 45 STEP 5
0050   PRINT DEGR,FNS(DEGR),FNC(DEGR)
0060 NEXT DEGR
0070 STOP

* TAB=12

* RUN
 0          0             1
 5          8.71556E-2    .996195
 10         .173648       .984808
 15         .258819       .965926
 20         .34202        .939693
 25         .422618       .906308
 30         .5            .866025
 35         .573576       .819152
 40         .642787       .766045
 45         .707107       .707107

STOP
AT 0070
*
```

Comment (2)

Shows the nesting of
functions.

3.6 DELAY

Format

DELAY = <expr>

<expr>: a numeric expression, which is evaluated to an integer in the range 0 < <expr> <= 60.

Use

As a statement to interrupt program execution for a specified number of seconds.

Remarks

1. <expr> is the number of seconds for which the program is stopped.

2. When <expr> seconds have passed since the DELAY statement was encountered, program execution will continue from the first statement following the DELAY statement.

3.7 DIM

Format

```
    |<svar>(<m>)             |  [ |<svar>(<m>)             | ]
DIM {<array>(<m>)            } [,{<array>(<m>)            }] ...
    |<array>(<row>,<col>)    |  [ |<array>(<row>,<col>)    | ]
```

<svar>: a string variable.

<m>: a numeric expression, which evaluates to the length of a string variable or the number of elements in a one-dimensional array.

<array>: an array name.

<row>: a numeric expression, which evaluates to the number of rows in an array.

<col>: a numeric expression, which evaluates to the number of columns in an array.

Use

As a statement or command to define explicitly the size of one or more numeric variable arrays or string variables. (For the dimensioning of string variables, see Chapter 5).

Remarks

1. Array elements

For the concept of arrays, see Chapter 2. The DIM statement is used to declare the size of an array to be a specified number of elements for each dimension, for example:

```
10 DIM A(13),B(7,7),C(20,5)
```

Until a value is assigned by the user's program, the value of all elements in an array is zero.

Any variable or expression that is used for a subscript must evaluate to a value in the range

```
1  <= value <= upper bound declared in the DIM statement
```

for example:

```
15 X=2
20 PRINT B(1,X↑2)
```

If the variable or expression subscript does not evaluate to an integer, RC BASIC will convert it using the INT(X) function (see Ch. 4).

If a subscript evaluates to an integer greater than the upper bound of the dimension for the array or less than 1, the error message 0031: SUBSCRIPT ERROR will be output.

2. Redimensioning arrays

One can redimension a previously defined array during execution of a program by declaring the array in another DIM statement. The total number of elements in the redimensioned array must not exceed the previous total number of elements, for example:

```
100 DIM A(3,3)
 .
 .
 .
200 DIM A(2,3)
 .
 .
 .
300 DIM A(2,2)
```

The values assigned to elements in the array A(3,3) are reassigned to elements in the array A(2,3) and then to elements in the array A(2,2):

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

```
A(1,1) = 1  A(1,1) = 1  A(1,1) = 1
A(1,2) = 2  A(1,2) = 2  A(1,2) = 2
A(1,3) = 3  A(1,3) = 3  A(2,1) = 3
A(2,1) = 4  A(2,1) = 4  A(2,2) = 4
A(2,2) = 5  A(2,2) = 5
A(2,3) = 6  A(2,3) = 6
A(3,1) = 7
A(3,2) = 8
A(3,3) = 9
```

3. Total number of elements

The total number of elements in an array may not exceed 32 767. Available memory will normally impose a limit, however.

3.8 END

Format

END [<comment>]

<comment>: a text comment.

Use

As a statement to terminate execution of the program and to return control to interactive mode.

Remarks

1. RC BASIC includes the END statement, but does not require its use to declare the physical end of a program. If control passes through the last executable statement of the program and if that statement does not change the flow of control, i.e. is not a GOTO or similar statement, then the program will transfer control to interactive mode.

2. Multiple END statements may appear in the same program, and when encountered will terminate execution of the program followed by a prompt (*) output on the user's terminal.

3. The word END may be followed by a comment.

Example

```
*20 PRINT "PROGRAM DONE"
*30 GOTO 60
   .
   .
   .
*50
*60 END
* RUN
PROGRAM DONE

END
AT 0060
*
```

3.9 ENTER

For description, see Chapter 9.

3.10 EXEC

Format

EXEC <name>

<name>: the name of a procedure. <name> may also be a simple numeric variable.

Use

As a statement to execute a procedure defined by PROC-ENDPROC (see Sect. 3.26).

Remarks

1. Rules

<name> is the name of the procedure to be executed. If <name> is a simple numeric variable, it may be assigned a value before the procedure is called; it may also be assigned a new value by the procedure before control is returned to the main program.

2. Program operation

a. When the EXEC statement is encountered, a search is made for the procedure named <name>.

b. If <name> is not found, the error message 0046: PROCEDURE DOES NOT EXIST will be output.

c. The statements in the procedure are executed until an ENDPROC or RETURN statement is encountered. Control is then returned to the first statement following the EXEC statement.

Example

See PROC-ENDPROC (Sect. 3.26).

3.11 FOR-NEXT

Format

```
FOR <control var> = <expr1>  TO  <expr2> [STEP <expr3>]
  <statements>
NEXT <control var>
```

<control var>: an unsubscripted numeric variable.

<expr1>: a numeric expression defining the first or initial value of <control var>.

<expr2>: a numeric expression defining the terminating value of <control var>.

<expr3>: a numeric expression defining the increment added to <control var> each time the loop is executed.

<statements>: a block of statements, which may also contain FOR-NEXT loops.

Use

As a statement to establish the initial, terminating, and incremental values of a control variable, which is used to determine the number of times a block of statements contained in a FOR-NEXT loop is to be executed. The loop is repeated until the value of the control variable meets the termination condition.

Remarks

1. Rules

   a. The control variable <control var> must not be subscripted.

   b. For every FOR or NEXT statement there must be a matching NEXT or FOR statement, otherwise an error message (0021: FOR WITHOUT NEXT or 0022: NEXT WITHOUT FOR) will be output.

   c. The expressions <expr1>, <expr2>, and <expr3> may have positive or negative values; <expr3> must not be zero.

d. If the STEP <expr3> argument is omitted in the FOR statement, <expr3> is assumed to be +1.

e. The termination condition for a FOR-NEXT loop depends on the values of <expr1> and <expr3>. The loop will terminate if <expr3> is positive and the next value of <control var> is greater than <expr2>, or if <expr3> is negative and the next value of <control var> is less than <expr2>.

Note: If the value of <expr1> (the initial value) meets the termination condition, <statements> will not be executed even once.

f. If the body of a FOR-NEXT loop is entered at any point other than the FOR statement, the error message 0022: NEXT WITHOUT FOR will be output when the NEXT statement corresponding to the skipped FOR statement is encountered.

g. When the termination condition is met, the loop is exited.

2. Program loop operation

a. <expr1>, <expr2>, and <expr3> are evaluated. If <expr3> is not specified, it is assumed to be +1.

b. <control var> is set equal to <expr1>.

c. If <expr3> is positive (negative) and <control var> is greater than (less than) <expr2>, the termination condition is satisfied and control passes to the first statement following the corresponding NEXT statement; otherwise step e is performed.

d. If <expr3> is positive (negative) and <control var> + <expr3> is greater than (less than) <expr2>, the termination condition is satisfied and control passes to the first statement following the corresponding NEXT statement; otherwise <control var> is set equal to <control var> + <expr3>.

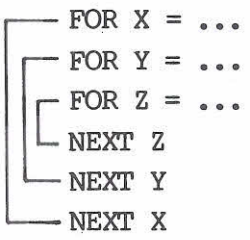
e. <statements> is executed.

f. Step d is repeated.

3. Nested loops

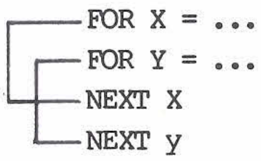
FOR-NEXT loops may be nested to a depth of seven. The FOR statement and its terminating NEXT statement must be completely contained within the loop in which they are nested, for example:

Legal nesting



```
FOR X = ...
FOR Y = ...
FOR Z = ...
NEXT Z
NEXT Y
NEXT X
```

Illegal nesting



```
FOR X = ...
FOR Y = ...
NEXT X
NEXT y
```

Example 1

Comment (1)

```
0010 FOR I=1 TO 25
0020   FOR J=1 TO 25 STEP 7
0030   NEXT J
0040 NEXT I
0050 PRINT I,J
0060 STOP
```

```
 25              22
```

Final values of I and J before their terminating values were exceeded.

Example 2

```
0010 LET A=10
0020 FOR I=A TO 1 STEP -1
0030   PRINT TAB(I);
0040   FOR J=A TO I STEP -1
0050     PRINT "**";
0060   NEXT J
0070   PRINT
0080   REM NEW LINE
0090 NEXT I
0100 STOP
```

```
01234567890123456789

          **
         ****
        ******
       ********
      **********
     ************
    **************
   ****************
  ******************
********************
```

Comment (2)

Shows nested FOR-NEXT loops.

Example 3

```
0010 DIM TEXT$(30)
0020 LET TEXT$="TEXT1TEXT2TEXT3TEXT4TEXT5TEXT6"
0030 FOR I=5 TO 0 STEP -1
0040   PRINT TEXT$(I*5+1,(I+1)*5)
0050 NEXT I
0060 STOP
```

```
TEXT6
TEXT5
TEXT4
TEXT3
TEXT2
TEXT1
```

Comment (3)

A string array can be implemented by means of a FOR-NEXT construction.

3.12 GOSUB and RETURN

Format

```
GOSUB <line no.>
 .
 .
 .
<statements>
RETURN [<comment>]
```

```
      <line no.>:  the first statement of a subroutine.
    <statements>:  a block of statements.
       <comment>:  a text comment.
```

Use

As a statement to direct program control to the first statement of a subroutine. RETURN exits the subroutine and returns control to the first statement following the GOSUB statement that caused the subroutine to be entered.

Remarks

1. A subroutine is a convenient means of executing the same block of statements at different places in a program. Subroutines may be nested to a depth of seven. Nesting occurs when a subroutine is called during the execution of another subroutine.

2. A subroutine may be entered only by means of a GOSUB statement, otherwise the error message 0019: RETURN WITHOUT GOSUB will be output when the RETURN statement is encountered.

3. A subroutine may contain more than one RETURN statement, should program logic require the subroutine to terminate at one of a number of different places.

4. Although a subroutine may appear anywhere in a program, it is good practice to place the subroutine distinctly separate from the main program. In order to prevent inadvertent entry of the subroutine by other than a GOSUB statement, the subroutine should be preceded by a STOP statement (see Sect. 3.33) or a GOTO statement (see Sect. 3.13) that directs control to a line number following the subroutine.

5. The word RETURN may be followed by a comment.

Example 1

```
0010 LET I=144
0020 GOSUB 0060
0030 LET I=169
0040 GOSUB 0060
0050 STOP
0060 PRINT "THE SQUARE ROOT OF";I;"IS:";SQR(I)
0070 RETURN
```

```
THE SQUARE ROOT OF 144 IS: 12
THE SQUARE ROOT OF 169 IS: 13
```

Example 2

```
0010 GOSUB 0040
0020 PRINT "EXAMPLE"
0030 GOTO 0140
0040 PRINT "NEST";
0050 GOSUB 0080
0060 PRINT "INE ";
0070 RETURN
0080 PRINT "ED ";
0090 GOSUB 0120
0100 PRINT "ROUT";
0110 RETURN
0120 PRINT "SUB";
0130 RETURN
0140 STOP
```

```
NESTED SUBROUTINE EXAMPLE
```

3.13 GOTO

Format

GOTO <line no.>

    <line no.>: a line number.

Use

As a statement to transfer control unconditionally to a statement that is not in normal sequential order.

Remarks

1. If control is transferred to an executable statement, that statement and those following it will be executed.

2. If control is transferred to a non-executable statement, such as DATA, program execution will continue at the first executable statement following the non-executable statement.

Example

```
0010 READ NUMBER
0020 PRINT NUMBER;
0030 IF NUMBER<>0 THEN GOTO 0010
0040 STOP
0050 DATA 10,9,8,7,6,5,4,3,2,1,0


 10  9  8  7  6  5  4  3  2  1  0
```

3.14 IF-THEN

Format

IF <expr> [THEN] <statement>

<expr>: an expression which, when evaluated, has the value true (<> 0) or false (= 0).

<statement>: any RC BASIC statement except CASE-WHEN-ENDCASE, DATA, DEF, END, FOR-NEXT, ENDIF, ELSE, PROC-ENDPROC, REM, REPEAT-UNTIL, and WHILE-ENDWHILE.

Use

As a statement or command to execute a single statement depending on whether the value of an expression is true or false.

Remarks

Program operation

1. If the value of <expr> is true (<> 0), <statement> is executed. If <statement> does not cause transfer of control to

another part of the program, execution will then continue at the first statement following the IF-THEN statement.

2. If the value of <expr> is false (= 0), <statement> is not executed.

   Note: Since the internal representation of non-integer numbers may not be exact (.2 cannot be represented exactly, for example), it is advisable to test for a range of values when testing for a non-integer. If, for example, the result of a computation, A, was to be 1.0, a reliable test for 1 would be

   ```
   IF ABS(A-1.0)<1.0E-6 THEN ...
   ```

   If this test succeeded, A would be equal to 1 to within 1 part in 10↑6. This is approximately the accuracy of single-precision floating-point calculations.

Example 1

```
0010 LET I=10
0020 IF I>5 THEN GOTO 0040
0030 PRINT "DON'T ENTER HERE"
0040 PRINT "PRINT THIS"
0050 STOP
```

Comment (1)

The statement GOTO 40 is executed only if I > 5.

```
PRINT THIS
```

Example 2a

```
0010 LET A=5; B=5
0020 PRINT "A AND B ARE";
0030 IF A<>B THEN PRINT " NOT";
0040 PRINT " EQUAL"
0050 STOP
```

```
A AND B ARE EQUAL
```

Example 2b

```
0010 LET A=5; B=7
0020 PRINT "A AND B ARE";
0030 IF A<>B THEN PRINT " NOT";
0040 PRINT " EQUAL"
0050 STOP

A AND B ARE NOT EQUAL
```

3.15 IF-THEN-ENDIF

Format

```
IF <expr> [THEN] [DO]
   <statements>
ENDIF [<comment>]
```

<expr>: an expression which, when evaluated, has the value true (<> 0) or false (= 0).

<statements>: a block of statements.

<comment>: a text comment.

Use

As a statement to execute a block of statements depending on whether the value of an expression is true or false.

Remarks

1. Rules

   a. For every IF-THEN/ENDIF statement there must be a matching ENDIF/IF-THEN statement.

   b. If <statements> is entered at any point other than the IF-THEN statement, the error message 0056: ENDIF WITHOUT IF will be output when the ENDIF statement is encountered.

2. Program operation

   a. If the value of <expr> is true (<> 0), <statements> is executed once.

b. Execution will then continue at the first statement following the ENDIF statement.

3. Nested constructions

IF-THEN-ENDIF/IF-THEN-ELSE-ENDIF constructions may be nested to a depth of seven.

4. The word ENDIF may be followed by a comment.

Example 1

```
0010 LET I=1
0020 IF I THEN
0030   PRINT "I<>0"
0040   LET I=I+1
0050 ENDIF
0060 PRINT "AFTER ENDIF, I=";I
0070 STOP
```

```
I<>0
AFTER ENDIF, I= 2
```

Comment (1)

The block of statements between IF-THEN and ENDIF is executed only if I is true (I <> 0).

Example 2

```
0010 LET I=0
0020 IF I THEN
0030   PRINT "DON'T ENTER HERE"
0040   LET I=I+1
0050 ENDIF
0060 PRINT "AFTER ENDIF, I=";I
0070 STOP
```

```
AFTER ENDIF, I= 0
```

Example 3

```
0010 LET I=26
0020 IF I/2>=13 THEN
0030   PRINT "SHOULD ENTER HERE"
0040 ENDIF
0050 STOP
```

```
SHOULD ENTER HERE
```

Example 4

```
0010 DIM NAME$(4)
0020 LET NAME$="JOHN"
0030 IF NAME$(2,3)="OH" THEN
0040   PRINT "NAMES CONTAINS 'OH' "
0050   LET NAME$(2,3)="  "
0060   PRINT NAME$
0070 ENDIF
0080 STOP

NAMES CONTAINS 'OH'
J  N
```

3.16 IF-THEN-ELSE-ENDIF

Format

```
IF <expr> [THEN] [DO]
  <statements-1>
ELSE [<comment>]
  <statements-2>
ENDIF [<comment>]
```

<expr>: an expression which, when evaluated, has the value true (<> 0) or false (= 0).

<statements-1>: a block of statements which is executed if the value of <expr> is true (<> 0).

<statements-2>: a block of statements which is executed if the value of <expr> is false (= 0).

<comment>: a text comment.

Use

As a statement to execute one of two blocks of statements depending on whether the value of an expression is true or false.

Remarks

1. Rules

   If <statements-1> or <statements-2> is entered at any point other than the IF-THEN/ELSE statement, an error message (0051: ELSE WITHOUT IF or 0056: ENDIF WITHOUT IF) will be output when the ELSE/ENDIF statement if encountered.

2. Program operation

   a. <expr> is evaluated.

   b. If the value of <expr> is true (<> 0), <statements-1> is executed.

   c. If the value of <expr> is false (= 0), <statements-2> is executed.

   d. When <statements-1> or <statements-2> has been executed and if neither has caused transfer of control to another part of the program, execution will continue at the first statement following the ENDIF statement.

3. Nested constructions

   IF-THEN-ENDIF/IF-THEN-ELSE-ENDIF constructions may be nested to a depth of seven.

4. The words ELSE and ENDIF may be followed by comments.

Example

```
0010 DIM PRICE(4),NUMBER(4)
0020 FOR ITEMNO=1 TO 4
0030   IF (ITEMNO=1) OR (ITEMNO=3) THEN
0040     LET PRICE(ITEMNO)=10
0050     LET NUMBER(ITEMNO)=7
0060   ELSE
0070     LET PRICE(ITEMNO)=25
0080     LET NUMBER(ITEMNO)=9
0090   ENDIF
0100 NEXT ITEMNO
0110 PRINT "ITEMNO","NUMBER","PRICE"
0120 FOR I=1 TO 4
0130   PRINT I,NUMBER(I),PRICE(I)
0140 NEXT I
0150 STOP
```

Comment

If ITEMNO = 1 or 3, the statements in lines 40 and 50 are executed; otherwise the statements in lines 70 and 80 are executed.

```
ITEMNO        NUMBER        PRICE
 1             7             10
 2             9             25
 3             7             10
 4             9             25
```

3.17 INPUT

Format

```
                  { <var> }[            { ,<var> }]
INPUT [<slit-0>,] { <svar>}[[,<slit-n>] { ,<svar>}] ...
```

<slit-0, slit-n>: string literals.
<var>: a numeric variable.
<svar>: a string variable.

Use

As a statement or command to assign values entered from the user's terminal during program execution to a list of numeric or string variables.

Remarks

1. When an INPUT statement is executed, the system outputs a question mark (?) as an initial prompt unless the INPUT

statement contains <slit-0>, in which case <slit-0> is output.

2. The user responds by typing a list of data items, each of which is separated from the next by a comma. The last item is followed by a carriage return.

3. Data items will be read as long as the arguments in the INPUT statement are <var> or <svar>. If a string literal, <slit-n>, is encountered in the argument list, <slit-n> will be output and any remaining items entered by the user will be skipped.

4. If the data list is terminated (by pressing the RETURN key) before values have been assigned to all of the variables in the argument list, the system will output a question mark as a prompt, indicating that further items are expected.

5. Data entered in response to a prompt must be of the same type (numeric or string) as the variable in the argument list for which the data is being supplied. Variables in the argument list may be subscripted or unsubscripted.

6. If the entered data does not match the type of a variable in the argument list, the system will output / ? in response to the erroneous input. The user can then enter data of the correct type.

7. A comma may not be used as a separator between string data items. These items must be separated either by a carriage return or, if typed on the same line, by a quotation mark (") followed by a comma.

Example 1

```
* LIST
0010 DIM NAME$(20),ADDRESS$(20)
0020 INPUT NAME$,ADDRESS$
0030 PRINT NAME$,ADDRESS$
0040 INPUT "NAME ",NAME$,"ADDRESS ",ADDRESS$
0050 PRINT NAME$,ADDRESS$

* RUN
 ? ROBERT CLARK )
 ? 9 MAIN STREET )
ROBERT CLARK 9 MAIN STREET              ← PRINT output.
NAME RAYMOND CLARKE )
ADDRESS 61 HIGHWAY )
RAYMOND CLARKE            61 HIGHWAY    ← PRINT output.

END
AT 0050
*
```

Comments (1)

The underlined texts are those entered by the user.

Example 2

```
* LIST
0010 INPUT N1,N2,N3,N4
0020 PRINT N1;N2;N3;N4

* RUN
 ? 1 )
 ? 2 )
 ? 3 )
 ? 4 )
 1  2  3  4

END
AT 0020
* RUN
 ? 5,6,7,8 )
 5  6  7  8

END
AT 0020
```

Comments (2)

The underlined texts are those entered by the user.

A question mark is output as a prompt until data has been supplied for all arguments.

All data items can be typed on a single line when separated by commas.

```
* RUN
 ? 9)
 ? U)
/ ? 8)
 ? 7)
 ? Y)
/ ? 6)
 9  8  7  6

END
AT 0020
*
```

The type of the entered data items must match the type of the arguments.

3.18 LET

Format

$$[\text{LET}] \begin{Bmatrix} \langle\text{var}\rangle \\ \langle\text{svar}\rangle \end{Bmatrix} = \langle\text{expr}\rangle \left[; \begin{Bmatrix} \langle\text{var}\rangle \\ \langle\text{svar}\rangle \end{Bmatrix} = \langle\text{expr}\rangle \right] \ldots$$

<var>: a numeric variable.
<svar>: a string variable.
<expr>: a numeric, relational, or string expression.

Use

As a statement or command to evaluate an expression and assign the resultant value to a variable.

Remarks

1. Use of the mnemonic LET is optional.

2. The variables may be subscripted.

3. Numeric or relational expressions may be assigned to numeric variables.

4. String expressions may be assigned to string variables.

Example 1

```
10 LET A=A+1
```

Comment (1)

The variable A is assigned a value one greater than it was before.

Example 2

```
20 A(2,1)=B↑2+10
```

Comment (2)

The element in row 2, column 1 of the array A is assigned the value of the expression B↑2+10.

Example 3

```
0010 LET I=3
0020 LET J=4
0030 LET K=I+J; L=I*J
0040 PRINT I,J,K,L
0050 STOP

 3          4          7          12
```

Comment (3)

One line may contain several assignments as shown in line 30.

3.19 NEW

For description, see Chapter 9.

3.20 ON-ERR

Format

ON ERR THEN <statement>

<statement>: any RC BASIC statement except CASE-WHEN-ENDCASE, DATA, DEF, END, FOR-NEXT, ENDIF, ELSE, ON, PROC-ENDPROC, REM, REPEAT-UNTIL, and WHILE-ENDWHILE.

Use

As a statement to enable the programmer to take special action, if an error occurs during program execution.

Remarks

1. Usually a program is interrupted and an error message output, if an error occurs during program execution. If an ON-ERR statement has been executed, however, a run-time error will cause <statement> to be executed.

2. The ON-ERR statement closely resembles the ON-ESC statement; see, therefore, Section 3.21, Remarks 2 to 5, for further details.

Example

```
0010 ON ERR THEN EXEC OUTERROR
0020 LET A=10/0
0030 LET B=D
0040 LET C=SYS(18)
0050 PROC OUTERROR
0060   PRINT "ERROR :";SYS(7)
0070   ON ERR THEN EXEC OUTERROR
0080 ENDPROC
```

Comment

If an error occurs, the error code will be output and execution will continue.

```
ERROR : 16                    0016:  ARITHMETIC ERROR
ERROR : 17                    0017:  UNDEFINED VARIABLE
ERROR : 34                    0034:  ILLEGAL FUNCTION
                                     ARGUMENT
```

3.21 ON-ESC

Format

ON ESC THEN <statement>

<statement> : any RC BASIC statement except CASE-WHEN-ENDCASE, DATA, DEF, END, FOR-NEXT, ENDIF, ELSE, ON, PROC-ENDPROC, REM, REPEAT-UNTIL, and WHILE-ENDWHILE.

Use

As a statement to enable the programmer to take special action, if the ESCape key is pressed during program execution.

Remarks

1. Usually a program is interrupted, if the ESCape key is pressed during program execution. If an ON-ESC statement has been executed, however, pressing the ESCape key will cause <statement> to be executed.

2. If the ESCape key has been pressed once and <statement> has been executed, the program will be interrupted if the ESCape key is pressed again, unless a new ON-ESC statement has been executed (see Example).

3. If <statement> is a GOSUB or EXEC statement, then when a RETURN or ENDPROC statement is encountered, control will be transferred to the statement following the last statement executed before the ESCape key was pressed.

4. ON-ESC statements may be placed anywhere in a program, so that different actions may be taken in different parts of the program.

5. Execution of the statement ON ESC THEN STOP will in all circumstances cause restoration of the normal ESCape key function.

```
Example                                Comment

* LIST
0010 ON ESC THEN GOSUB 0060
0020 FOR I=1 TO 20
0030   PRINT I;
0040 NEXT I
0050 STOP
0060 PRINT
0070 PRINT "ESCAPE PRESSED, I =  ";I
0080 RETURN

* RUN
 1  2  3  4  5
ESCAPE PRESSED, I =  6
 7  8  9  10  11  1                    ←When ESC is pressed a
STOP                                   second time, the program
AT 0030                                is interrupted.

* LIST
0010 ON ESC THEN GOSUB 0060
0020 FOR I=1 TO 20
0030   PRINT I;
0040 NEXT I
0050 STOP
0060 PRINT
0070 PRINT "ESCAPE PRESSED, I =  ";I
0075 ON ESC THEN GOSUB 0060            ←This statement has been
0080 RETURN                            inserted in the program
                                       above.
* RUN
 1  2  3  4  5
ESCAPE PRESSED, I =  5
 6  7  8  9  1
ESCAPE PRESSED, I =  10
 11  12  13  14  15  16
ESCAPE PRESSED, I =  16
 17  18  19  20
STOP
AT 0050
*
```

3.22 ON-GOTO/GOSUB

Format

```
                       { GOTO  }
ON <expr> [THEN] { GOSUB } <line no.> [,<line no.>] ...
```

<expr>: a numeric expression which is evaluated to an integer.

<line no.>: a line number in the current program. The positions of line numbers in the argument list are numbered in sequence from 1 to n.

Use

As a statement to transfer control to one of several lines in a program depending on the computed value of an expression when the ON statement is executed.

Remarks

1. <expr> is evaluated. If it is not an integer, the fractional part is ignored.

2. The program transfers control to the line whose sequence number in the argument list corresponds to the computed value of <expr>.

3. If <expr> evaluates to an integer that is greater than the sequence number of the last line number in the argument list or that is less than or equal to zero, the ON statement is ignored and control passes to the next statement.

4. The ON-GOSUB statement must contain a list of line numbers each of which is the first line of a subroutine within the current program.

Example	Comment
10 ON M-5 GOTO 500,75,1000	If M-5 evaluates to 1, 2, or 3, control will be transferred to line number 500, 75, or 1000, respectively. If M-5 evaluates to any other value, control will be transferred to the next sequential statement in the program.

3.23 PAGE

For description, see Chapter 9.

3.24 PRINT

Format

$$\begin{Bmatrix} ; \\ \text{PRINT} \end{Bmatrix} \left[\begin{Bmatrix} \langle\text{expr}\rangle \\ \langle\text{slit}\rangle \\ \langle\text{svar}\rangle \end{Bmatrix} \left[\begin{Bmatrix} , \\ ; \end{Bmatrix} \begin{Bmatrix} \langle\text{expr}\rangle \\ \langle\text{slit}\rangle \\ \langle\text{svar}\rangle \end{Bmatrix} \right] \ldots \right] \left[\begin{Bmatrix} , \\ ; \end{Bmatrix} \right]$$

<expr>: a numeric or relational expression.

<slit>: a string literal.

<svar>: a string variable.

Use

As a statement or command to perform any of the following output operations at the user's terminal:

1. Print variables and constants (numeric or string).

2. Print the result of a computation.

3. Print a combination of 1 and 2.

4. Print a blank line (skip a line).

Remarks

1. Printing numbers

Numbers (integer, decimal, or E-type) are printed in the following form:

<sign><number><space>

The sign is either minus (-) or a blank space for plus; the number is always followed by a blank space (see Ch. 2).

2. Zone spacing of output

The print line on a terminal is divided into print zones. The width of a print zone can be set by means of the TAB command (see Ch. 9). The default zone spacing is 14 columns. This spacing is used in the examples below. The first column on a print line is column number 0.

0 13	14 27	28 41	42 55	56 69
← 14 →	← 14 →	← 14 →	← 14 →	← 14 →
columns	columns	columns	columns	columns

A comma (,) between items in the argument list causes the next print element to be output starting from the leftmost position of the next zone. If there are no more zones on the current line, printing continues in the first zone on the next line. If a print element requires more than one zone, the next element is printed in the next free zone (see Exx. 1 and 3).

Before each print element is output, its length is compared with the space remaining on the current line. If the space is insufficient, the element is moved to the next line. If the length of an element is greater than the length of the print line (see the PAGE command, Ch. 9), the error message 0036: PRINT ELEMENT TOO LONG is output.

3. Compact spacing of output

A semicolon (;) between items in the argument list causes the next print element to be output starting from the next character position.

Note: A blank space is always printed after a number. A blank space is also reserved for the plus (+) sign, even though the sign is not printed (see Ex. 2).

4. Spacing to the next line

When the last element in a PRINT statement argument list has been printed, a carriage return and line feed are output unless this last element is followed by a comma or a semicolon, in which case the carriage return and line feed are suppressed and the elements in the next PRINT statement argument list are printed on the same line in accordance with the comma or semicolon punctuation (see Exx. 2 and 3).

5. Printing blank lines

A PRINT statement with no arguments or punctuation causes a carriage return and line feed to be output (see Ex. 4).

6. Additional printing versatility

See the TAB(X) function (Sect. 3.35), the PAGE and TAB commands (Ch. 9), and the PRINT USING statement (Sect. 3.25).

Example 1

```
0010 LET X=25
0020 PRINT "THE SQUARE ROOT OF X IS:",SQR(X)
0030 STOP
```

```
012345678901234567890123456789012345678901
|             |             |             |
THE SQUARE ROOT OF X IS:     5
```

Comment (1)

If a print element requires more than one zone, the next element is printed in the next free zone.

Example 2

```
0010 LET X=5
0020 PRINT X;X↑X;X*X;X↑2;
0030 PRINT SQR(X);SQR(X)↑2
0040 PRINT -X;(-X)↑3
0050 STOP
```

```
012345678901234567890123456789012345678901
|             |             |             |
 5  3125  25  25  2.23607  5
-5 -125
```

Comment (2)

Shows the use of the semicolon as a spacing character. The semicolon terminating line 20 suppresses a carriage return and line feed.

Example 3

```
0010 LET X=100
0020 PRINT X,X↑2,SQR(X),
0030 PRINT X↑3
0040 PRINT "END"
0050 STOP
```

Comment (3)

Shows the use of the comma as a spacing character. The comma terminating line 20 suppresses a carriage return and line feed.

```
01234567890123456789012345678901234567890123456789012345
|             |             |             |             |
 100           10000         10            1E+6
END
```

Example 4

```
0010 LET X=10
0020 PRINT X,
0030 PRINT X↑2,
0040 PRINT
0050 PRINT X,
0060 PRINT X↑2
0070 PRINT "END"
0080 STOP
```

```
0123456789012345678901234567
|             |             |
 10            100
 10            100
END
```

Comment (4)

The PRINT statement in line 40 causes a carriage return and line feed to be output.

Example 5

```
0010 DIM A$(10)
0020 LET A$="ABCDE"
0030 PRINT A$="ABCDE";A$<"ABCDEF";A$>"B";1=1;2>5
0040 STOP

TRUE  TRUE  FALSE TRUE  FALSE
```

Comment (5)

If a print element is a relational expression, either the word TRUE or the word FALSE will be printed.

3.25 PRINT USING

Format

$$\text{PRINT USING } \langle\text{format}\rangle,\ \left\{\begin{array}{l}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{array}\right\}\left[\left\{\begin{array}{c},\\ ;\end{array}\right\}\left\{\begin{array}{l}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{array}\right\}\right]\ \ldots\ \left[\left\{\begin{array}{c},\\ ;\end{array}\right\}\right]$$

<format>: a string literal or string variable that specifies the format (see Remarks) for printing the items in the argument list.

<expr>: a numeric or relational expression.

<slit>: a string literal.

<svar>: a string variable.

Use

As a statement to output the values of items in the argument list using a specified format.

Remarks

1. The occurrence of a separator other than a comma, e.g. a semicolon or TAB, between items in the argument list will cause the remaining items to be printed as if no format had been specified (see the PRINT statement, Sect. 3.24).

2. The PRINT USING statement is executed as a PRINT statement (see Sect. 3.24); however, if the system does not include the PRINT USING facility, execution of a PRINT USING statement will cause the string <format> to be printed, followed by the items in the argument list.

3. The <format> expression may have more than one <format>

field and may include string literals as well as the following special characters, which are used to format numeric output:

. + - $, ↑

Note: The character Δ is used in the following descriptions to indicate a blank space.

a. Digital representation (#)

For each # in the <format> field, a digit (0-9) is substituted from the <expr> argument.

<format>	<expr>	Output	Comments
#####	25	ΔΔΔ25	The digits are right justified in the field with leading blanks.
#####	-30	ΔΔΔ30	Signs and other non-digits are ignored.
#####	1.95	ΔΔΔΔ2	Only integers are represented; the number is rounded to an integer.
#####	598745	*****	If the number in <expr> has more digits than specified by <format>, a field of all asterisks is output.

b. Decimal point (.)

The decimal character (.) places a decimal point within the string of digits in the fixed position in which it appears in <format>. Digit (#) positions which follow the decimal point are filled; no blank spaces are left in these digit positions. When <expr> contains more fractional digits than <format> allows, the fraction will be rounded to the limits of <format>. When <expr> contains less fractional digits than specified by <format> , zeroes are output to fill the positions.

<format>	<expr>	Output	Comments
#####.##	20	ΔΔΔ20.00	Fractional positions are filled with zeroes.
#####.##	29.347	ΔΔΔ29.35	Rounding occurs on fractions.
#####.##	789012.34	********	When <expr> has too many significant digits to the left of a decimal point, a field of all asterisks, including the decimal point, is output.

c. Fixed sign (+ or -)

A fixed sign character appears as a single plus (+) or minus (-) sign either in the first character position or in the last character position in the <format> field.

A fixed plus (+) sign prints the sign (+ or -) of <expr> in the position in which the fixed plus (+) sign is placed in <format>.

A fixed minus (-) sign prints a minus (-) sign for negative values of <expr> or a blank space for positive values of <expr> in the position in which the fixed minus (-) sign is placed in <format>.

When a fixed sign is used, any leading zeroes appearing in <expr> will be replaced by blanks, except for a single leading zero preceding a decimal point.

<format>	<expr>	Output	Comments
+##.##	20.5	+20.50	
+##.##	1.01	+Δ1.01	Blanks precede the number.
+##.##	-1.236	-Δ1.24	
+##.##	-234.0	******	
###.##-	20.5	Δ20.50Δ	
###.##-	000.01	ΔΔ0.01Δ	One leading zero before the decimal point is printed.
###.##-	-1.236	ΔΔ1.24-	
###.##-	-234.0	234.00-	

d. Floating sign (++ or --)

A floating sign appears as two or more plus (++) or minus (--) signs at the beginning of the <format> field.

A floating plus (++) sign outputs a plus or minus sign immediately before the value of <expr> with no separating blank spaces as would occur with a fixed sign.

A floating minus (--) sign outputs either a minus or a blank (for plus) immediately preceding the value.

Positions occupied in <format> by the second sign and any additional signs can be used for numeric positions in the value of <expr>.

<format>	<expr>	Output	Comments
---.##	-20	-20.00	The second and third minus signs are treated as # on output.
---.##	-200	******	Too many digits to the left of the decimal point.
---.##	2	ΔΔ2.00	

Note that <format> may include a floating sign (++ or --) or a floating dollar sign ($$), as described below, but not both.

e. Fixed dollar sign ($)

A fixed dollar sign appears as a single dollar ($) sign in either the first or the second character position in the <format> field, causing a dollar ($) sign to be output in that position. If the dollar sign ($) is in the second position, it must be preceded by a fixed sign (+ or -).

A fixed dollar sign ($) causes leading zeroes in the value of <expr> to be replaced by blanks.

<format>	<expr>	Output	Comments
-$###.##	30.512	Δ$Δ30.51	
$###.##+	-30.512	$Δ30.51-	

f. Floating dollar sign ($$)

A floating dollar sign appears as two or more dollar ($$) signs beginning in either the first or the second character position in the <format> field. If the dollar signs ($$) start in the second position, they must be preceded by a fixed sign (+ or -).

A floating dollar sign ($$) causes a dollar ($) sign to be placed immediately before the first digit of the <expr> value.

Note that <format> may include a floating dollar sign ($$) or a floating sign (++ or --), as described above, but not both.

<format>	<expr>	Output	Comments
+$$$#.##	13.20	+ΔΔ$13.20	Additional dollar signs may be replaced by #, as with floating signs (++ and --).
$$##.##-	-1.0	Δ$01.00-	Leading zeroes are not suppressed in the # part of the field.

g. Separator (,)

The comma separator (,) places a comma in the fixed position in which it appears in a string of digits in the <format> field.

If a comma would be output in a field of suppressed leading zeroes (blanks), a blank space is output in the position for the comma.

<format>	<expr>	Output	Comments
+$#,###.##	30.6	+$ΔΔΔ30.60	A blank space is output for the comma.
+$#,###.##	2000	+$2,000.00	
++##,###	00033	Δ+00,033	A comma is printed when leading zeroes are not suppressed.

h. Exponent indicator (↑↑↑↑)

Four consecutive up-arrows (↑↑↑↑) are used to indicate an exponent field in <format>. The four arrows will be output as E+nn, where each n is a digit.

If the exponent field in <format> does not contain exactly four up-arrows, a run-time error will result.

<format>	<expr>	Output	Comments
+##.##↑↑↑↑	170.35	+17.03E+01	
+##.##↑↑↑↑	-.2	-20.00E-02	
++##.##↑↑↑↑	6002.35	+600.24E+01	

4. A <format> expression, as previously remarked, may have more than one <format> field and may include string literals as well as the special formatting characters just described. Values of items in the argument list of the PRINT USING statement are sequentially assigned to <format> fields.

   RC BASIC distinguishes string literals from <format> fields by the characters that appear in the latter, for example:

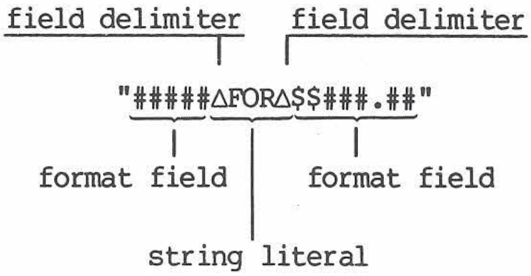
"TWO FOR $1.25"	$1.25 is part of the string literal.
"TWO FOR $$$.##"	$$$.## is a <format> field in the <format> expression.
"ANSWER IS -85"	-85 are characters of a string literal.
"ANSWER IS -###"	-### is a <format> field in the <format> expression.

5. A <format> expression may be specified by referencing a previously defined string variable, for example:

```
15 DIM S$(10)
20 LET S$="##.##"
30 PRINT USING S$,1.5,2
```

6. <format> fields in a <format> expression are delimited by the use of a non-formatting character before or after the <format>

field, for example:



7. String literals may appear in the argument list of the PRINT USING statement and will be superimposed on a <format> field in the following manner:

   a. Each character of the string literal replaces a single <format> field character, which may be any of the special formatting characters, i.e. #, decimal point, +, -, $, comma, and ↑.

   b. Strings are left justified in the <format> field, and filled with spaces, if necessary.

   c. If the number of characters in the string is greater than the number of characters in the <format> field, the string will be truncated to fit the field, for example:

```
5 PRINT USING "###,###.##","TEST","CHARACTER","SEVENTY-FIVE"
RUN
TEST△△△△△△CHARACTER△SEVENTY-FI
```

8. When there are more items in the argument list than <format> fields in the <format> expression, the <format> fields will be used repetitively. Thus, for example, in

   "####△@$###.##△PER△###"

   the first, fourth, seventh, etc. items will be formatted using the <format> field ####; the second, fifth, eighth, etc. items using the <format> field $###.##; and the third, sixth, ninth, etc. items using the <format> field ###. The embedded blank spaces, @ sign, and PER are string literals and delimit the <format> fields.

Examples | Comments

```
 .
 .
 .
100 PRINT USING "A(#)Δ=Δ##.#",I,A(I)
 .
 .
 .
RUN
A(1)Δ=Δ17.9
```

Possible output includes two <format> fields and two string literals.

```
 .
 .
 .
100 PRINT USING "###.##Δ",I,A,B
 .
 .
 .
RUN
ΔΔ1.00ΔΔ17.90ΔΔ25.77Δ
```

Possible output with the <format> expression repeated for each item in the argument list.

3.26 PROC-ENDPROC

Format

```
PROC <name>
  <statements>
ENDPROC [<comment>]
```

<name>: the name of a procedure. <name> may also be a simple numeric variable.

<statements>: a block of statements.

<comment>: a text comment.

Use

As a statement to define a procedure which can be called by means of an EXEC statement (see Sect. 3.10).

Remarks

1. A procedure is a convenient means of executing the same block of statements at different places in a program. Procedures may be nested to a depth of seven. Nesting occurs when a procedure is called during the execution of another procedure.

2. Rules

   a. For every PROC/ENDPROC statement there must be one and only one ENDPROC/PROC statement.

   b. A procedure may be placed anywhere in the program. When a PROC statement is encountered, a search is made for the corresponding ENDPROC statement, and program execution continues from the first statement following this ENDPROC statement.

   c. If the body of a procedure is entered without use of an EXEC statement, the error message 0019: RETURN WITHOUT GOSUB will be output when the ENDPROC statement is encountered.

   d. A procedure may contain one or more RETURN statements. When encountered, RETURN has the effect of an ENDPROC statement.

   e. If the name of the procedure, <name>, is a simple numeric variable, it may be assigned a value before the procedure is called; it may also be assigned a new value by the procedure before control is returned to the main program.

3. Program operation

   a. When the procedure is called, execution starts at the first statement following the PROC statement.

   b. Execution continues until a RETURN or ENDPROC statement is encountered; after this, control is passed to the first statement following the EXEC statement that called the procedure.

4. The word ENDPROC may be followed by a comment.

Example

```
0010 PROC GCD
0020   REM THE PROCEDURE FINDS THE GREATEST COMMON DIVISOR IN A AND B
0030   PRINT "GCD IN";A;"AND";B;":",
0040   WHILE A<>B DO
0050     IF A>B THEN
0060       LET A=A-B
0070     ELSE
0075       LET B=B-A
0080     ENDIF
0090   ENDWHILE
0100   LET GCD=A
0110   REM A AND B DESTROYED
0120 ENDPROC
0130 REM
0140 REM MAIN PROGRAM, A AND B ARE READ FROM THE TERMINAL
0150 REM
0160 INPUT A,B
0170 IF (A<=0) OR (B<=0) THEN STOP
0180 EXEC GCD
0190 PRINT GCD
0200 GOTO 0160
```

```
GCD IN 1 AND 1 :              1
GCD IN 2 AND 4 :              2
GCD IN 24 AND 68 :            4
GCD IN 24 AND 16 :            8
GCD IN 16 AND 24 :            8
GCD IN 345 AND 27 :           3
GCD IN 345 AND 344 :          1
GCD IN 6 AND 11 :             1
GCD IN 11 AND 66 :            11
GCD IN 1 AND 100 :            1
GCD IN 56 AND 7 :             7
GCD IN 56 AND 8 :             8
```

3.27 RANDOMIZE

Format

```
RANDOMIZE
```

Use

As a statement or command to cause the random number generator to start at a different point in the sequence of random numbers generated by the RND(X) function (see Ch. 4).

Remarks

1. Normally, the same sequence of random numbers is generated by successive use of the RND(X) function. This feature is useful in debugging programs. If, when the program has been found to run successfully, different starting points in the sequence are desired, the RANDOMIZE statement should be included in the program before the first occurrence of an RND(X) function.

2. The RANDOMIZE statement resets the random number generator based on the time of day, thereby producing different random numbers each time a program using the RND(X) function is run.

Example

```
* LIST
0010 LET I=0
0015 RANDOMIZE
0020 REPEAT
0030   PRINT RND(I);
0040   LET I=I+1
0050 UNTIL I=4
0060 STOP
0070 GOTO 0010

* RUN
 .921699  .341465  .710697  .816505
STOP
AT 0060
* RUN
 .249755  .980187  .616137  .890037
STOP
AT 0060
* CON
 .159747  .605283  .118429  .322262
STOP
AT 0060
*
```

Comment

This program produces different results each time it is executed.

3.28 READ

Format

$$\mathrm{READ}\begin{Bmatrix}\langle\mathrm{var}\rangle\\ \langle\mathrm{svar}\rangle\end{Bmatrix}\left[\begin{Bmatrix},\langle\mathrm{var}\rangle\\ ,\langle\mathrm{svar}\rangle\end{Bmatrix}\right]\ldots$$

<var>: a numeric variable.
<svar>: a string variable.

Use

As a statement or command to read in values from the list defined by one or more DATA statements and to assign the values to the variables listed in the READ statement.

Remarks

1. READ statements are always used in conjunction with DATA statements (see Sect. 3.4).

2. The variables listed in the READ statement may be subscripted or unsubscripted numeric or string variables.

3. The order in which variables appear in the READ statement is the order in which values for the variables are retrieved from the DATA statement list.

4. A data element pointer is moved to the next available value in the DATA statement list as values are retrieved for variables in the READ statement. If the number of variables in the READ statement exceeds the number of values in the DATA statement list, the error message 0015: NO MORE DATA FOR READ is output.

5. The type (numeric or string) of the READ statement variable must match the type of the corresponding data element value, otherwise the error message 0066: TYPE CONFLICT is output.

6. The RESTORE statement (see Sect. 3.31) can be used to reset the data element pointer to the first item of the lowest numbered DATA statement or the first item of a particular DATA statement.

Example

```
0010 DIM TEXT$(20),A(10)
0020 READ TEXT$,NUMBER
0030 DATA "READ DATA",25,1,2
0040 READ A(2),A(4)
0050 PRINT "TEXT","NUMBER","A(2)","A(4)"
0060 PRINT TEXT$,NUMBER,A(2),A(4)
0070 STOP
```

```
TEXT          NUMBER        A(2)          A(4)
READ DATA      25            1             2
```

3.29 REM

Format

REM [<comment>]

<comment>: a text comment.

Use

As a statement to insert explanatory comments within a program.

Remarks

1. REM statements are non-executable, but are stored with the program and output exactly as entered when LISTed (see LIST, Ch. 9).

2. If control is transferred to a REM statement from a GOTO or GOSUB statement, execution continues from the next executable statement following the REM statement. If no executable statement follows the REM statement, the program will act as if an END statement (see Sect. 3.8) had been encountered and control will return to interactive mode.

3. Optional text comments may also be inserted after the words ENDCASE, END, RETURN, ENDIF, ELSE, ENDPROC, REPEAT, STOP, and ENDWHILE, for example:
   ENDWHILE END OF SEARCH

Example

```
10 REM REMARKS THROUGHOUT A PROGRAM CAN
20 REM HELP EXPLAIN THE PURPOSE OF STATEMENTS.
30 REM LINES 10, 20, and 30 ARE NOT EXECUTED.
```

3.30 REPEAT-UNTIL

Format

```
REPEAT [<comment>]
  <statements>
UNTIL <expr>
```

```
     <comment>:  a text comment.
  <statements>:  a block of statements.
        <expr>:  a relational expression.
```

Use

As a statement to execute a block of statements repetitively until the value of an expression is true.

Remarks

1. Rules

   a. <expr> is a relational expression whose value is either true or false, e.g. NAME$ = "JOHN", I > 25.

   b. If the body of a REPEAT-UNTIL loop is entered at any point other than the REPEAT statement, the error message 0058: UNTIL WITHOUT REPEAT will be output when the UNTIL statement corresponding to the skipped REPEAT statement is encountered.

2. Program loop operation

   a. <statements> is executed.

   b. <expr> is evaluated.

   c. If the value of <expr> is false, step a is repeated.

   d. If the value of <expr> is true, the termination condition is satisfied and control passes to the first statement following the corresponding UNTIL statement.

Note: <statements> is always executed at least once.

3. Nested loops

REPEAT-UNTIL loops may be nested to a depth of seven.

4. The word REPEAT may be followed by a comment.

Example 1

```
0010 LET I=1
0020 REPEAT
0030   PRINT I;
0040   LET I=I+1
0050 UNTIL I>10
0060 PRINT "<13><10>AFTER 'UNTIL' , I=";I
0070 STOP
```

```
 1  2  3  4  5  6  7  8  9  10
AFTER 'UNTIL' , I= 11
```

Comment (1)

The block of statements between REPEAT and UNTIL is repeated until I > 10.

Example 2

```
0010 LET I=20
0020 REPEAT
0030   PRINT "EXECUTED ONCE"
0040   LET I=I-1
0050 UNTIL I>10
0060 PRINT "AFTER 'UNTIL' , I=";I
0070 STOP
```

```
EXECUTED ONCE
AFTER 'UNTIL' , I= 19
```

Comment (2)

The block of statements between REPEAT and UNTIL is always executed at least once.

Example 3

```
0010 LET A=10; B=1
0020 REPEAT
0030   PRINT
0040   PRINT "A=";A,
0050   REPEAT
0060     PRINT "B=";B;
0070     LET B=B+1
0080   UNTIL B=5
0090   LET B=1; A=A-1
0100 UNTIL A<7
0110 PRINT
0120 PRINT "AFTER LAST 'UNTIL', A,B=";A;B
0130 STOP
```

Comment (3)

Shows nested REPEAT-UNTIL loops.

```
A= 10         B= 1 B= 2 B= 3 B= 4
A= 9          B= 1 B= 2 B= 3 B= 4
A= 8          B= 1 B= 2 B= 3 B= 4
A= 7          B= 1 B= 2 B= 3 B= 4
AFTER LAST 'UNTIL', A,B= 6  1
```

3.31 RESTORE

Format

RESTORE [<line no.>]

<line no.>: a DATA statement line number.

Use

As a statement or command to reset the data element pointer either to the beginning of the DATA statement list or to a particular DATA statement.

Remarks

1. If the RESTORE statement is used without an argument, the data element pointer is reset to the beginning of the DATA statement list, i.e. to the first item in the lowest numbered DATA statement (see Sect. 3.4).

2. If the RESTORE statement contains an argument, the data element pointer is reset to the first item in the DATA

statement specified by <line no>.

3. If <line no.> does not exist in the program, the data element pointer is reset to the beginning of the DATA statement list.

Example

```
0010 DATA 1,2,3,4
0020 READ I,J,K,L
0030 RESTORE
0040 REM RESET TO BEGINNING
0050 READ M,N
0060 RESTORE 0100
0070 REM RESET TO LINE 100
0080 READ O,P,Q,R
0090 PRINT I;J;K;L;M;N;O;P;Q;R
0100 DATA 5,6,7,8
0110 STOP
```

```
 1  2  3  4  1  2  5  6  7  8
```

Comment

One can choose among several DATA statements by means of RESTORE. In line 30, the data element pointer is reset to the beginning of the DATA statement list.

3.32 SAVE

For description, see Chapter 9.

3.33 STOP

Format

```
STOP [<comment>]

      <comment>:  a text comment.
```

Use

As a statement to terminate execution of the current program and to return control to interactive mode.

Remarks

1. STOP statements may be placed anywhere in the program. When STOP is encountered, the system will terminate execution and

output the following on the user's terminal:

```
STOP
AT <xxxx>
*
```

<xxxx>: the line number of the STOP statement.

2. After control has returned to interactive mode, the program may be restarted from the first line number (see RUN, Ch. 9) or continued in its current state (see CON or RUN <line no.>, Ch. 9).

3. The word STOP may be followed by a comment.

Example

```
* LIST
0010 REM TERMINATE PROGRAM BY STOP
0020 INPUT A
0030 IF A<0 THEN STOP
0040 GOTO 0020

* RUN
 ? 1
 ? 3
 ? -5

STOP
AT 0030
*
```

3.34 TAB

For description, see Chapter 9.

3.35 TAB(X) function

Format

```
TAB(<expr>)
```

<expr>: an expression which is evaluated to an integer.

Use

As a function in PRINT statements (see Sect. 3.24) to tabulate the printing position for an item in the argument list to the column number evaluated from an expression.

Remarks

1. As the print line columns are numbered from 0, the position indicated by the TAB(X) function is always relative to 0, e.g. the column number indicated by TAB(31) is 30.

2. A PRINT statement may contain several TAB(X) functions, each of which affects only the item in the argument list that immediately follows it. The printing position for this item will depend on the value of <expr> and the punctuation (; or ,) following TAB(X).

3. If <expr> evaluates to a column number greater than or equal to the current column number and less than the length of the print line, the value of <expr> indicates the new column position.

   If TAB(X) is followed by a semicolon (;), the new column position remains unchanged (see Example).

   If TAB(X) is followed by a comma (,), the new column position is changed to the leftmost position of the next print zone unless the new column position coincides with the leftmost position of a print zone (see Example).

   After determination of the new column position, the item in the argument list immediately following the TAB(X) function is printed (see Remarks in Sect. 3.24).

4. If <expr> evaluates to a column number less than the current column number, the TAB(X) function is ignored and positioning proceeds as in 3.

5. If <expr> evaluates to a column number greater than the length of the print line, <expr> is reduced modulo the length of the print line and positioning proceeds as in 3.

   The length of the print line (width of the page) can be set by means of the PAGE command (see Ch. 9).

6. If <expr> evaluates to 0, e.g. TAB(0), a carriage return and line feed are output and positioning proceeds as in 3.

Example

```
0010 LET POS=5; NUMBER=-1048
0020 PRINT TAB(POS);NUMBER;TAB(7*POS);NUMBER
0030 PRINT TAB(POS),NUMBER,TAB(7*POS),NUMBER
0040 PRINT TAB(31);NUMBER
0050 STOP
```

```
01234567890123456789012345678901234567890123456789012345
|             |             |             |            |
     -1048                         -1048
               -1048                       -1048
                               -1048
```

Comment

Shows the use of the semicolon and the comma as spacing characters in conjunction with the TAB(X) function.

3.36 WHILE-ENDWHILE

Format

```
WHILE <expr> [THEN] DO
  <statements>
ENDWHILE [<comment>]
```

```
          <expr>:  a relational expression.
    <statements>:  a block of statements.
       <comment>:  a text comment.
```

Use

As a statement to execute a block of statements repetitively while the value of an expression is true.

Remarks

1. Rules

   a. <expr> is a relational expression whose value is either true or false, e.g. I <= 10, MONTH < 13.

b. For every WHILE statement there must be a matching ENDWHILE statement, otherwise the error message 0053: WHILE WITHOUT ENDWHILE is output.

c. If the body of a WHILE-ENDWHILE loop is entered at any point other than the WHILE statement, the error message 0054: ENDWHILE WITHOUT WHILE will be output when the ENDWHILE statement corresponding to the skipped WHILE statement is encountered.

2. Program loop operation

a. <expr> is evaluatd.

b. If the value of <expr> is false, the termination condition is satisfied and step e is performed.

c. <statements> is executed.

d. Step a is repeated.

e. Control passes to the first statement following the corresponding ENDWHILE statement.

Note: If the value of <expr> is false the first time the WHILE statement is encountered, <statements> is not executed even once.

3. Nested loops

WHILE-ENDWHILE loops may be nested to a depth of seven.

4. The word ENDWHILE may be followed by a comment.

Example 1

```
0010 LET I=1
0020 WHILE I<10 DO
0030   PRINT I;
0040   LET I=I+1
0050 ENDWHILE
0060 PRINT "<13><10>AFTER 'ENDWHILE' "
0070 STOP
```

```
 1  2  3  4  5  6  7  8  9
AFTER 'ENDWHILE'
```

Example 2

```
0010 LET I=11
0020 WHILE I<10 DO
0030   PRINT "DO NOT ENTER HERE"
0040   LET I=I-1
0050 ENDWHILE
0060 PRINT "AFTER 'ENDWHILE' "
0070 STOP
```

```
AFTER 'ENDWHILE'
```

Comment (2)

If <expr> is false when WHILE is encountered for the first time, <statements> is not executed even once.

Example 3

```
0010 LET I=1
0020 WHILE I<5 DO
0030   LET J=8
0040   PRINT "I=";I,
0050   WHILE J>I DO
0060     PRINT " J=";J;
0070     LET J=J-1
0080   ENDWHILE
0090   PRINT
0100   LET I=I+1
0110 ENDWHILE
0120 PRINT "AFTER LAST 'ENDWHILE' "
0130 STOP
```

```
I= 1           J= 8  J= 7  J= 6  J= 5  J= 4  J= 3  J= 2
I= 2           J= 8  J= 7  J= 6  J= 5  J= 4  J= 3
I= 3           J= 8  J= 7  J= 6  J= 5  J= 4
I= 4           J= 8  J= 7  J= 6  J= 5
AFTER LAST 'ENDWHILE'
```

Comment (3)

Shows nested WHILE-ENDWHILE loops.

4 RC BASIC Functions

4.1 Introduction

RC BASIC provides a number of functions to perform various calculations, thereby eliminating the need to write programs for these calculations.

RC BASIC functions generally have a three-letter mnemonic name, followed by a parenthesized expression as an argument, and may be used as an expression or included as part of an expression.

Standard mathematical functions included in RC BASIC and the values which they produce are as follows:

ABS(X)	Absolute value of X.
ATN(X)	Arctangent of X in radians.
COS(X)	Cosine of X, where X is in radians.
EXP(X)	e^X (-178 <= X <= 175).
LOG(X)	Natural logarithm of X (X > 0).
SIN(X)	Sine of X, where X is in radians.
SQR(X)	Square root of X (X >= 0).
TAN(X)	Tangent of X, where X is in radians.

In addition to these standard arithmetical and trigonometrical functions, the RC BASIC system includes the following:

FNa(d)	Functions defined by the user.
INT(X)	Integer value of X.
RND(X)	Random number between 0 and 1.
SGN(X)	Algebraic sign of X.
SYS(X)	System information functions.

Further RC BASIC functions, which are described in other chapters, are these:

CHR(X)	String function (see Ch. 5).
DET(X)	Matrix function (see Ch. 6).
EOF(X)	File function (see Ch. 8).
LEN(X$)	String function (see Ch. 5).
ORD(X$)	String function (see Ch. 5).
TAB(X)	Printing function (see Ch. 3).

4.2 ABS(X)

Format

ABS(<expr>)

<expr>: a numeric expression.

Use

As a function to return the absolute (positive) value of <expr>.

Example

```
* LIST
0010 PRINT ABS(-10);ABS(10)

* RUN
 10  10

END
AT 0010
*
```

4.3 ATN(X)

Format

ATN(<expr>)

<expr>: a numeric expression ($-\pi/2 <=$ ATN(<expr>) $<= \pi/2$).

Use

As a function to calculate the angle, in radians, whose tangent is <expr>.

Example

```
* LIST
0010 PRINT ATN(0);ATN(1)*180/SYS(14)

* RUN
 0  45

END
AT 0010
*
```

4.4 COS(X)

Format

COS(<expr>)

    <expr>: a numeric expression specified in radians.

Use

As a function to calculate the cosine of an angle which is expressed in radians.

Example

```
* LIST
0010 PRINT COS(0);COS(45*SYS(14)/180)

* RUN
 1  .707107

END
AT 0010
*
```

4.5 EXP(X)

Format

EXP(<expr>)

    <expr>: a numeric expression (-178 <= <expr> <= 175).

Use

As a function to calculate the value of e (2.71828) to the power of <expr>.

Example

```
* LIST
0010 PRINT EXP(1);EXP(2);EXP(2.5)

* RUN
 2.71828  7.38905  12.1825

END
AT 0010
*
```

4.6 FNa(d)

For description, see the DEF statement, Chapter 3.

4.7 INT(X)

Format

INT(<expr>)

<expr>: a numeric expression.

Use

As a function to return the value of the nearest integer not greater than <expr>.

Example

```
* LIST
0010 PRINT INT(4.567);INT(-4.567)

* RUN
 4 -5

END
AT 0010
*
```

4.8 LOG(X)

Format

LOG(<expr>)

<expr>: a numeric expression.

Use

As a function to calculate the natural logarithm of <expr>.

Example

```
* LIST
0010 PRINT LOG(2);LOG(EXP(1))

* RUN
 .693149  1

END
AT 0010
*
```

4.9 RND(X)

Format

RND(<expr>)

<expr>: a numeric expression (required, but not used).

Use

As a function to produce a pseudo random number, n, such that 0 <= n <1.

Remarks

1. The RND(X) function requires an argument, although the argument does not affect the resulting random number and the function does not affect the argument.

2. Each time the RND(X) function is called, it produces a pseudo random number in the range 0 to 1. The sequence in which these numbers are generated is fixed. The length of the sequence is 2↑16.

3. As the sequence of random numbers is fixed and the starting point in the sequence is reset to the same point each time a NEW or RUN command (see Ch. 9) is given, the sequence of numbers generated by the RND(X) function is reproducible.

4. The RANDOMIZE statement (see Ch. 3) causes the random number generator to start at a different point in the sequence of random numbers generated by the RND(X) function.

5. Each occurrence of the RND(X) function yields the value of the next random number in the list.

Example 1

```
* LIST
0010 LET I=0
0020 REPEAT
0030   PRINT RND(I);
0040   LET I=I+1
0050 UNTIL I=4
0060 STOP
0070 GOTO 0010
```

```
* RUN
 .21132   .14464   .852625   .927054
STOP
AT 0060
* RUN
 .21132   .14464   .852625   .927054
STOP
AT 0060
* CON
 .162866   .433095   .563933   .20965
STOP
AT 0060
*
```

Comment (1)

The RUN command resets the sequence of random numbers; the CON command does not.

Example 2

```
* LIST
0010 LET I=0
0020 WHILE I<4 DO
0030   PRINT INT(25*RND(I));
0040   LET I=I+1
0050 ENDWHILE
```

```
* RUN
 5  3  21  23
END
AT 0050
*
```

Comment (2)

The program produces random integers in the range 0 to 24.

4.10 SGN(X)

Format

SGN(<expr>)

    <expr>: a numeric expression.

Use

As a function to return a +1 if <expr> is greater than 0, a 0 if <expr> equals 0, and a -1 if <expr> is less than 0.

Example

```
* LIST
0010 PRINT SGN(-5);SGN(0);SGN(5)

* RUN
-1  0  1

END
AT 0010
*
```

Comment

Note that SGN(0) = 0.

4.11 SIN(X)

Format

SIN(<expr>)

    <expr>: a numeric expression specified in radians.

Use

As a function to calculate the sine of an angle which is expressed in radians.

Example

```
* LIST
0010 PRINT SIN(0);SIN(45*SYS(14)/180)

* RUN
 0  .707107

END
AT 0010
*
```

4.12 SQR(X)

Format

```
SQR(<expr>)
```

<expr>: a positive numeric expression.

Use

As a function to compute the square root of <expr>.

Example

```
* LIST
0010 PRINT SQR(25);SQR(25.734)

* RUN
 5  5.07287

END
AT 0010
*
```

4.13 SYS(X)

Format

SYS(<expr>)

<expr>: a numeric expression.

Use

As a function to return system information based on the value of <expr>, which is evaluated to an integer (0 to 15), as follows:

Function	Information	
SYS(0)	Time of day (seconds past midnight).	
SYS(1)	Day of the month (1 to 31).	Current date.
SYS(2)	Month of the year (1 to 12).	
SYS(3)	Year as two digits (e.g. 77).	
SYS(4)	Terminal port number (32, if console).	
SYS(5)	Time used in seconds since the terminal was logged on.	
SYS(6)	Number of file I/O statements executed (i.e. every statement or command referring to a file, e.g. PRINT FILE, LIST "$LPT").	
SYS(7)	Error code of the last run-time error.	
SYS(8)	File number of the file most recently referenced in a file I/O statement.	
SYS(9)	Page size (length of the print line).	
SYS(10)	Tab size (width of the print zone).	
SYS(11)	Hour of the day.	Current time of day.
SYS(12)	Minutes past the last hour.	
SYS(13)	Seconds past the last minute.	

SYS(14)   Constant π (3.14159).

SYS(15)   Constant e (2.71828).

4.14 TAN(X)

Format

TAN(<expr>)

<expr>: a numeric expression specified in radians.

Use

As a function to calculate the tangent of an angle which is expressed in radians.

Example

```
* LIST
0010 PRINT TAN(0);TAN(45*SYS(14)/180)

* RUN
 0  .999999

END
AT 0010
*
```

5 String Information

5.1 String concept

This chapter explains how strings are used in RC BASIC. The string concept is described in the present section, while three useful string functions, viz. CHR(X), LEN(X$), and ORD(X$) are described in the remaining sections.

5.1.1 String literals

A string is a sequence of characters, which may include letters, digits, spaces, and special characters. A string literal (string constant) is a string enclosed within quotation marks. String literals are often used in PRINT and INPUT statements (see Ch. 3), for example:

```
100 PRINT "THIS IS A STRING LITERAL"
200 INPUT "X=",X
```

The enclosing quotation marks are not printed when the string is output. Non-printing and special characters may be included in string literals by enclosing the decimal value of the character within angle brackets (<>), for example:

```
10 PRINT "USE DECIMAL 60 TO PRINT <60> IN STRINGS"
* RUN
USE DECIMAL 60 TO PRINT < IN STRINGS
```

The decimal values of all ASCII characters are given in Appendix D.

5.1.2 String variables

RC BASIC permits the use of string variables as well as string literals. A string variable name consists of a letter, followed by from 0 to 7 letters or digits, followed by a dollar sign ($), for example:

```
ANSWER$, TEXT$
```

String values are assigned to string variables, as described below, by means of LET, READ, and INPUT statements, for example:

```
INPUT ANSWER$
LET TEXT$="THIS IS A TEXT"
```

5.1.3 Dimensioning string variables

All string variables must be dimensioned before they are used. By dimensioning the variable, the user sets an upper bound for the number of characters that can be stored in it. Dimensioning is accomplished by means of the DIM statement (see Ch. 3), for example:

```
DIM ANSWER$(20)
DIM TEXT$(15),STRING$(5)
```

If the user attempts to assign a string literal that is too long to a string variable, the string literal will be truncated, for example:

```
10 DIM TEXT$(6)
20 LET TEXT$="LONG STRING"
30 PRINT TEXT$
* RUN
LONG S
```

A string may be of any length, the sole limitation being available memory.

5.1.4 Substrings

One can also reference a portion of a string variable. The general form of such substrings is the following:

$$\langle\text{svar}\rangle\left(\begin{Bmatrix}\langle\text{i}\rangle\\ \langle\text{j,k}\rangle\end{Bmatrix}\right)$$

<svar>: a string variable name.

<i>: a numeric expression indicating that the ith character in <svar> is to be referenced.

<j,k>: numeric expressions indicating that the jth through the kth characters in <svar> are to be referenced.

Example 1 shows how substrings can be referenced.

Example 1

```
0010 DIM TEXT$(20)
0020 LET TEXT$="AAAAAAAAAAAAAAAAAAAA"
0030 PRINT TEXT$
0040 LET TEXT$(5)="B"
0050 PRINT TEXT$
0060 LET TEXT$(10,13)="BCDE"
0070 PRINT TEXT$
0080 LET TEXT$(15,17)="BCDEFG"
0090 PRINT TEXT$
0100 PRINT TEXT$(1),TEXT$(8,13),TEXT$(20)
```

```
AAAAAAAAAAAAAAAAAAAA
AAAABAAAAAAAAAAAAAAA
AAAABAAAABCDEAAAAAAA
AAAABAAAABCDEABCDAAA
A             AABCDE        A
```

Note: When a value is to be assigned to a substring (i,j), a value must first be assigned to the substring (l,i-l). This can be done by initializing the entire string with blanks.

5.1.5 Assigning values to string variables

Values can be assigned to string variables by means of LET, READ and DATA, and INPUT statements (see Ch. 3) and READ FILE and INPUT FILE statements (see Ch. 8). Example 2 shows various uses of LET, READ and DATA, and INPUT.

5.1.6 Concatenation of strings

Any number of strings (variables or literals) can be concatenated by means of a LET statement having the following syntax:

$$[\text{LET}]\ \langle\text{svar}\rangle = \begin{Bmatrix}\langle\text{svar}\rangle\\ \langle\text{slit}\rangle\end{Bmatrix}\left[,\begin{Bmatrix}\langle\text{svar}\rangle\\ \langle\text{slit}\rangle\end{Bmatrix}\right]\dots$$

<svar>: a string variable.
<slit>: a string literal.

See Example 2, lines 50, 80, and 90.

Example 2

```
* LIST
0010 DATA "THIS"," A"
0020 DIM TEXT$(23),TEMP$(5)
0040 READ TEXT$(1,4)
0050 LET TEXT$=TEXT$," IS"
0060 READ TEXT$(LEN(TEXT$)+1,20)
0070 INPUT TEMP$
0080 LET TEXT$=TEXT$," ",TEMP$
0090 LET TEXT$=TEXT$," TEXT"
0100 PRINT TEXT$

* RUN
 ? SHORT)
THIS IS A SHORT TEXT

END
AT 0100
* RUN
 ? LONG)
THIS IS A LONG TEXT

END
AT 0100
*
```

Comment (2)

For LEN(X$), see Sect. 5.3.

The underlined texts are those entered by the user.

Note: When a value is assigned to a substring, the number of characters in the source string must not be less than the number of characters in the substring referenced; otherwise, the rest of the characters in the substring and any remaining characters in the string will be truncated. Example 3 provides an illustration of this.

```
Example 3                                Comment (3)

* LIST
0010 DIM TEXT$(10)
0020 LET TEXT$="AAAAAAAAAA"              The underlined
0030 PRINT TEXT$                         texts are those
0040 INPUT TEXT$(3,7)                    entered by the user.
0050 PRINT TEXT$

* RUN
AAAAAAAAAA
 ? 123↵                                  ←Source string.
AA123                                    ←TEXT$(6,10) has
                                         been truncated.
* LIST
0010 DIM TEXT$(10),TEMP$(10)             The desired result
0020 LET TEXT$="AAAAAAAAAA"              can be achieved in
0030 PRINT TEXT$                         this way.
0040 INPUT TEMP$
0050 LET TEMP$=TEMP$,"    "
0060 LET TEXT$(3,7)=TEMP$
0070 PRINT TEXT$

* RUN
AAAAAAAAAA
 ? 123↵
AA123  AAA
```

5.1.7 Relational string expressions

Strings (literals and variables) may be compared. The result of a comparison is either true or false. The strings are compared character by character, on the basis of their decimal values (see App. D), until a difference is found or the end of one or both strings is met.

If a character in a given position in one string has a higher decimal value than the character in the corresponding position in the other string, the first string is the greater of the two.

If the characters in corresponding positions are identical, but one string contains more characters than the other, the shorter string is the lesser of the two. Thus, for example, the expression "ABC" < "ABCD" is true.

Example 1

```
IF NAME$(1)="J" THEN
  .
  .
  .
```

```
Example 2                               Comment (2)

CASE ANSWER$="YES" OF
  .
  .
  .
WHEN 1                                  1 corresponds to true.
  .
  .
  .
WHEN 0                                  0 corresponds to false.
  .
  .
  .
ENDCASE
```

5.2 CHR(X) function

Format

```
CHR(<expr>)

        <expr>: a numeric expression.
```

Use

As a function to return the character corresponding to the number specified in the argument.

Remarks

1. The correspondence between numbers (decimal values) and characters is shown in Appendix D.

2. The number is found as <expr> modulo 128.

3. The CHR(X) function may be used in any string expression.

Example Comment

```
0010 TAB=4
0020 PAGE=32
0030 FOR I=65 TO 74
0040   PRINT I,CHR(I),
0050 NEXT I
```

```
 65 A    66 B    67 C    68 D
 69 E    70 F    71 G    72 H
 73 I    74 J
```

The characters corresponding to the numbers 65-74 are A-J.

5.3 LEN(X$) function

Format

```
     {<svar>}
LEN( {<slit>} )
```

<svar>: a string variable.
<slit>: a string literal.

Use

As a function to return the current length (number of characters) of the string specified in the argument.

Remarks

1. The LEN(X$) function may be used in any numeric expression.

2. If the string argument is empty, the value returned is 0.

Example Comment

```
0010 DIM TEXT$(10)
0020 TAB=12
0030 PRINT TEXT$,LEN(TEXT$)
0040 FOR I=1 TO 10
0050   LET TEXT$=TEXT$,CHR(I+64)
0060   PRINT TEXT$,LEN(TEXT$)
0070 NEXT I
```

For CHR(X), see Sect. 5.2.

```
              0
A             1
AB            2
ABC           3
ABCD          4
ABCDE         5
ABCDEF        6
ABCDEFG       7
ABCDEFGH      8
ABCDEFGHI     9
ABCDEFGHIJ    10
```

5.4 ORD(X$) function

Format

```
    ⎧<svar>⎫
ORD(⎩<slit>⎭)

      <svar>: a string variable.
      <slit>: a string literal.
```

Use

As a function to return the number of the first character of the string specified in the argument.

Remarks

1. The number returned is the character's decimal value, which is equivalent to its internal representation. (This number is also the ordinal number of the character; thus the character A, for example, which has the decimal value 65, is the 65th character in the ASCII character set). The correspondence between numbers and characters is shown in Appendix D.

2. The ORD(X$) function may be used in any numeric expression.

Example

Comment

```
0010 TAB=4
0020 PAGE=32
0030 DIM A$(10)
0040 LET A$="0123456789"
0050 FOR I=1 TO 10
0060   PRINT ORD(A$(I));A$(I),
0070 NEXT I
```

```
 48 0    49 1    50 2    51 3
 52 4    53 5    54 6    55 7
 56 8    57 9
```

The characters 0-9 have the internal decimal values 48-57.

6 Matrix Manipulation

6.1 Matrix operations

RC BASIC includes a special set of statements which allows the user to manipulate two-dimensional arrays (see Ch. 2) as matrices. Among the available matrix operations are the following:

Addition, subtraction, and multiplication.

Scalar multiplication.

Formation of a zero matrix (all elements set to 0).

Formation of a constant matrix (all elements set to 1).

Formation of an identity matrix (major diagonal elements set to 1, remaining elements set to 0).

Calculation of the inverse of a matrix.

Calculation of the determinant of a matrix.

Transposition of a matrix.

Input/output of a matrix via the user's terminal.

All of the above operations are described in the present chapter, while matrix file input/output statements are described in Chapter 8.

All statements involving matrix operations are introduced by the reserved word MAT.

6.2 Dimensioning matrices

A matrix is dimensioned as a two-dimensional array by means of the DIM statement (see Ch. 3). Thus the statement

```
10 DIM MATRIXA(10,20)
```

defines a matrix variable named MATRIXA, which has 10 rows and

20 columns (or a total of 10 x 20 = 200 elements).

A previously dimensioned matrix may be redimensioned by means of a new DIM statement, provided that the total number of elements does not exceed the previously declared total number of elements.

Matrix elements are stored by rows in ascending memory locations.

As a one-dimensional array containing i elements is considered a 1 x i matrix, the matrix operations described in the following sections also apply to one-dimensional arrays.

6.3 Matrix assignment statement

Format

```
MAT <mvar1> = <mvar2>

      <mvar1, mvar2>:  matrix variables.
```

Use

As a statement or command to copy the elements of one matrix to another matrix.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. The number of elements in <mvar2> must not exceed the number of elements in <mvar1>.

3. After the assignment, <mvar1> will have the same dimensions and values as <mvar2>.

Example   Comment

```
0010 DIM MATA(3,2),MATB(2,3)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050   NEXT J
0060 NEXT I
0070 PRINT "MATA : "
0080 MAT PRINT MATA
0090 PRINT "<13><10>MATB : "
0100 MAT PRINT MATB
0110 MAT MATB=MATA
0120 PRINT "<13><10>NEW MATB : "
0130 MAT PRINT MATB
```

```
MATA :
 11             12
 21             22
 31             32

MATB :
 0              0              0
 0              0              0

NEW MATB :
 11             12
 21             22
 31             32
```

Note that the dimensions of MATB have been changed.

6.4 Matrix addition/subtraction statement

Format

MAT <mvar1> = <mvar2> $\left\{\begin{matrix}+\\-\end{matrix}\right\}$ <mvar3>

<mvar1, mvar2, mvar3> : matrix variables.

Use

As a statement or command to perform the scalar addition or subtraction of two matrices.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. <mvar2> and <mvar3> must have the same dimensions.

3. The number of elements in <mvar2> (and <mvar3>) must not exceed the number of elements in <mvar1>.

4. The arithmetic is performed element by element with the resultant value assigned to the element in <mvar1>.

5. <mvar1> may appear on both sides of the equal sign.

6. After the addition or subtraction, <mvar1> will have the same dimensions as <mvar2> (and <mvar3>).

Example   Comment

```
0010 DIM MATA(3,2),MATB(3,2)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050     LET MATB(I,J)=2*(I*10+J)
0060   NEXT J
0070 NEXT I
0080 PRINT "MATA :  "
0090 MAT PRINT MATA
0100 PRINT "<13><10>MATB :  "
0110 MAT PRINT MATB
0120 MAT MATB=MATA+MATB                MATB appears on both sides
0130 PRINT "<13><10>MATA + MATB :"     of the equal sign.
0140 MAT PRINT MATB
```

```
MATA :
 11          12
 21          22
 31          32

MATB :
 22          24
 42          44
 62          64

MATA + MATB :
 33          36
 63          66
 93          96
```

The addition of MATA and MATB is performed element by element.

6.5 Matrix multiplication statement

Format

```
              { <mvar2>  }
MAT <mvar1> = { (<expr>) }  * <mvar3>
```

<mvar1, mvar2, mvar3>: matrix variables.
<expr>: a numeric expression (parenthesized).

Use

As a statement or command to perform the multiplication of one matrix either by another matrix or by a scalar (the value of a numeric expression).

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. The number of columns in <mvar2> must match the number of rows in <mvar3>.

3. If <mvar2> is an n x p matrix (i.e. with n rows and p columns) and <mvar3> is a p x m matrix, then <mvar1> will be an n x m matrix.

4. <mvar1> may not appear on the right-hand side of the equal sign, unless it is a scalar multiplication.

5. <mvar2> and <mvar3> may represent the same matrix (i.e. a square matrix, in which the number of rows matches the number of columns).

6. The product of <mvar2> and <mvar3> is calculated as follows:

   Each row of <mvar2> is multiplied by each column of <mvar3> such that the corresponding elements are multiplied and their products added together to provide the resultant value assigned to the element in <mvar1> (see Example).

   Row number i of <mvar2> * column number j of <mvar3> will, accordingly, result in element number (i,j) of <mvar1>.

7. If a matrix is multiplied by a scalar, each element in <mvar3> is multiplied by <expr> to give the corresponding element value in <mvar1>. In this case, the number of elements in <mvar3> must not exceed the number of elements in <mvar1>. After the multiplication, <mvar1> will have the same dimensions as <mvar3>.

Example

```
0010 DIM MATA(3,2),MATB(2,3),MATC(3,3)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050     LET MATB(J,I)=I*20+J
0060   NEXT J
0070 NEXT I
0080 PRINT "MATA : "
0090 MAT PRINT MATA
0100 PRINT "<13><10>MATB : "
0110 MAT PRINT MATB
0120 MAT MATC=MATA*MATB
0130 PRINT "<13><10>MATA * MATB :"
0140 MAT PRINT MATC
```

```
MATA :                                          Comment
 11          12
 21          22
 31          32

MATB :
 21          41          61
 22          42          62

MATA * MATB :
 495         955         1415
 925         1785        2645
 1355        2615        3875
```

$$495 = (11 \quad 12) \times \begin{pmatrix} 21 \\ 22 \end{pmatrix}$$

$$= (11 \times 21) + (12 \times 22)$$

6.6 DET(X) function

Format

```
<var> = DET(<expr>)

       <var>:  a numeric variable.
      <expr>:  a numeric expression (required, but not used).
```

Use

As a function to return the determinant of the last matrix inverted by a MAT INV statement (see Sect. 6.10).

Remarks

1. The DET(X) function requires an argument, although the argument does not affect the resulting determinant and the function does not affect the argument.

2. For calculation of the determinant of a matrix, see Section 6.10.

Example

See the MAT INV statement (Sect. 6.10).

6.7 MAT CON statement

Format

```
MAT <mvar> = CON

      <mvar>:  a matrix variable.
```

Use

As a statement or command to initialize a matrix such that all elements are set to one.

Remarks

1. The matrix must have been dimensioned before the statement is executed.

2. All elements of <mvar> are set to one regardless of any previously assigned values.

3. The resulting matrix is often called a constant matrix.

Example

```
0010 DIM MATA(3,2)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050   NEXT J
0060 NEXT I
0070 PRINT "MATA :  "
0080 MAT PRINT MATA
0090 MAT MATA=CON
0100 PRINT "<13><10>NEW MATA :"
0110 MAT PRINT MATA

MATA :
 11             12
 21             22
 31             32

NEW MATA :
 1              1
 1              1
 1              1
```

6.8 MAT IDN statement

Format

MAT <mvar> = IDN

<mvar>: a matrix variable.

Use

As a statement or command to initialize a matrix such that all elements (i,i) are set to one and the remaining elements are set to zero.

Remarks

1. The matrix must have been dimensioned before the statement is executed.

2. If <mvar> is an n x p matrix (i.e. with n rows and p

columns), then all elements (i,i), 1 <= i <= minimum (n,p), are set to one and the remaining elements are set to zero (see Example).

3. If <mvar> is a square matrix (i.e. where n = p), then the resulting matrix will be the identity matrix, in which all elements of the major diagonal are set to one and the remaining elements are set to zero (see Example).

Example

```
0010 DIM MATA(3,2),MATB(6,6)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050   NEXT J
0060 NEXT I
0070 PRINT "MATA :  "
0080 MAT PRINT MATA
0090 MAT MATA=IDN
0100 PRINT "<13><10>NEW MATA :"
0110 MAT PRINT MATA
0120 MAT MATB=IDN
0130 PRINT "<13><10>MATB :"
0140 MAT PRINT MATB;
```

```
MATA :
 11             12
 21             22
 31             32

NEW MATA :
 1              0
 0              1
 0              0

MATB :
 1  0  0  0  0  0
 0  1  0  0  0  0
 0  0  1  0  0  0
 0  0  0  1  0  0
 0  0  0  0  1  0
 0  0  0  0  0  1
```

6.9 MAT INPUT statement

Format

```
MAT INPUT <mvar1> [,<mvar2>, ... ,<mvar-n>]

      <mvar1, mvar2, mvar-n>: matrix variables.
```

Use

As a statement or command to assign numeric values entered from the user's terminal during program execution to the elements of one or more matrices.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. When a MAT INPUT statement is executed, the system outputs a question mark (?) as an initial prompt.

3. The user responds by typing a list of numeric data items, each of which is separated from the next by a comma. The last item is followed by a carriage return.

4. If the data list is terminated (by pressing the RETURN key) before values have been assigned to all of the matrix elements, the system will output / ? as a prompt, indicating that further items are expected.

6.10 MAT INV statement

Format

```
MAT <mvar1> = INV(<mvar2>)

      <mvar1, mvar2>:  matrix variables.
```

Use

As a statement or command to invert a matrix and assign the resultant element values to another matrix.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. The inverse, B, of a matrix, A, is defined such that the products of A x B and B x A are both equal to the identity matrix.

3. <mvar2> must be a square matrix.

4. The determinant (see Remark 6) of <mvar2> must not equal zero.

5. After the inversion of a matrix, the determinant of that matrix can be obtained by means of the DET(X) function (see Sect. 6.6).

6. Matrix determinants

   In order to calculate the inverse of a matrix, one must first calculate the determinant of that matrix.

   The calculation of the determinant of a 2 x 2 matrix is described in the following. For larger matrices, consult a mathematics text.

   The determinant of a 2 x 2 matrix

   $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is written as $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$.

   The determinant is the scalar

   (a x d) - (b x c)

   i.e. multiplication along the diagonals and subtraction of the product of the second diagonal from the product of the major diagonal.

6.11 MAT PRINT statement

Format

$$\text{MAT PRINT <mvar>} \left[\begin{Bmatrix} ; \\ , \end{Bmatrix} \text{<mvar>}\right] \ldots \; [;]$$

<mvar>: a matrix variable.

Use

As a statement or command to output the values of the elements of one or more matrices on the user's terminal.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. If a semicolon is used after a matrix variable in the argument list, the elements of that matrix will be output with compact spacing, i.e. each element will be output starting from the next character position. (Note: A blank space is always printed after an element and a blank space is reserved for the plus sign, even though this sign is not printed).

3. If a comma or carriage return is used after a matrix variable in the argument list, the elements of that matrix will be output with zone spacing, i.e. each element will be output starting from the leftmost position of a print zone. The width of the print zones can be set by means of the TAB command (see Ch. 9).

4. The values of the elements are output by rows in ascending order. If a row cannot be contained on a single print line, a carriage return and line feed are output and the row is continued on the next print line.

Example   Comments

```
0010 TAB=10
0020 PAGE=72
0030 DIM MATA(3,2),MATB(2,3),ARRAY(10)
0040 FOR I=1 TO 3
0050   FOR J=1 TO 2
0060     LET MATA(I,J)=I*10+J
0070     LET MATB(J,I)=J*10+I
0080   NEXT J
0090 NEXT I
0100 PRINT "<13><10>"
0110 MAT PRINT MATA,MATB
0120 PRINT "<13><10>"
0130 MAT PRINT MATA;MATB;
0140 PRINT "<13><10>"
0150 MAT PRINT MATA,MATB;
0155 TAB=5
0160 PRINT "<13><10>"
0170 MAT PRINT ARRAY
0180 MAT ARRAY=CON
0190 PRINT "<13><10>"
0200 MAT PRINT ARRAY;
```

```
 11        12                                     Statement 110.
 21        22                                     Zone spacing
 31        32                                     is used (comma
                                                  or carriage
 11        12        13                           return as sepa-
 21        22        23                           rator).

 11  12                                           Statement 130.
 21  22                                           Compact spacing
 31  32                                           is used (semi-
                                                  colon as sepa-
 11  12  13                                       rator).
 21  22  23

 11        12                                     Statement 150.
 21        22
 31        32

 11  12  13
 21  22  23

 0    0    0    0    0    0    0    0    0    0   Statement 170.

 1  1  1  1  1  1  1  1  1  1                     Statement 200.
```

6.12 MAT READ statement

Format

```
MAT READ <mvar> [,<mvar>] ...

      <mvar>:  a matrix variable.
```

Use

As a statement or command to read in numeric values from the list defined by one or more DATA statements (see Ch. 3) and to assign the values to the elements of one or more matrices.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. Values are assigned to matrix elements by rows in ascending order.

Example

```
0010 DIM MATA(3,2),MATB(2,3)
0020 DATA 1,2,3,4,5,6,7,8,9
0030 MAT READ MATA
0040 PRINT "MATA :"
0050 MAT PRINT MATA
0060 RESTORE
0070 MAT READ MATB
0080 PRINT "<13><10>MATB :"
0090 MAT PRINT MATB

MATA :
 1          2
 3          4
 5          6
MATB :
 1          2          3
 4          5          6
```

6.13 MAT TRN statement

Format

```
MAT <mvar1> = TRN(<mvar2>)

      <mvar1, mvar2>:  matrix variables.
```

Use

As a statement or command to transpose a matrix and assign the resultant element values to another matrix.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. A matrix is transposed by reversing the row and column assignments of the matrix elements, i.e. row number 1 in <mvar2> becomes column number 1 in <mvar1> and column number 1 in <mvar2> becomes row number 1 in <mvar1>.

3. If <mvar2> is an n x m matrix, <mvar1> will be an m x n matrix.

4. <mvar1> and <mvar2> must be two distinct matrices.

6.14 MAT ZER statement

Format

```
MAT <mvar> = ZER

      <mvar>:  a matrix variable.
```

Use

As a statement or command to initialize a matrix such that all elements are set to zero.

Remarks

1. The matrix must have been dimensioned before the statement is executed.

2. All elements of <mvar> are set to zero regardless of any previously assigned values.

3. The resulting matrix is often called a zero matrix.

Example

```
0010 DIM MATA(3,2)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050   NEXT J
0060 NEXT I
0070 PRINT "MATA : "
0080 MAT PRINT MATA
0090 MAT MATA=ZER
0100 PRINT "<13><10>NEW MATA :"
0110 MAT PRINT MATA
```

```
MATA :
 11            12
 21            22
 31            32

NEW MATA :
 0             0
 0             0
 0             0
```

7 Logical Discs and Related Commands

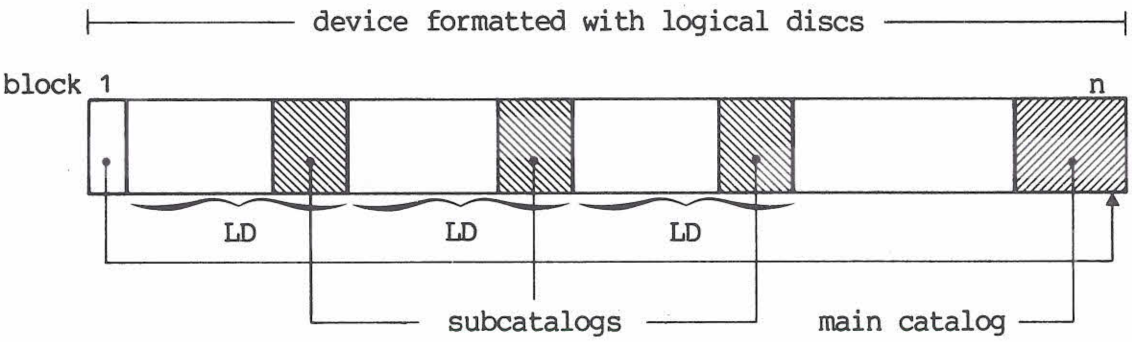
7.1 Introduction

An RC BASIC disc file (see the introduction to Chapter 8) comprises a number of consecutive blocks in a logical disc.

Any system device may in principle contain logical discs, but in practice only such read/write devices as flexible discs and moving-head disc files are used for this purpose. (Note that here a moving-head disc file is considered a device).

Logical discs are created in a device by means of a special formatting program (described in the separate publication RC BASIC System Logical Disc Formatting Program Operating Guide).

A device, i.e. a flexible disc or moving-head disc file, containing logical discs (LD's) is structured as follows:



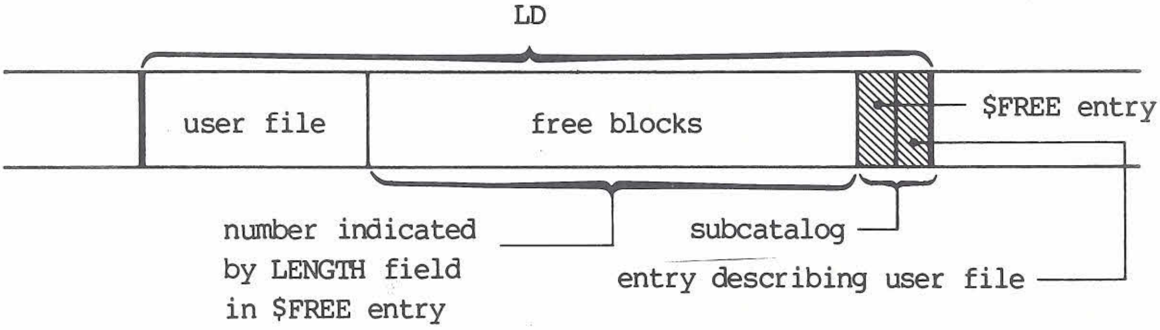
The main catalog describes the logical discs contained in the device. Each main catalog entry has the following structure:

USERS	Number of logical disc users.
KEY	Protection key of the logical disc.
	Not used.
LENGTH	Number of blocks allocated to the logical disc.
NAME	Name of the logical disc (1 to 8 characters).

The subcatalog in each logical disc describes the files contained in that logical disc. Each subcatalog entry has the following structure:

USERS	Number of file users.
LAST BLOCK	Last block written in the file (sequential access) or number of records in the file (random access).
LAST BYTE	Last byte written in the last block (sequential access) or record length in bytes (random access).
LENGTH	Number of blocks allocated to the file.
NAME	Name of the file (1 to 8 characters).

A subcatalog always contains an entry in which NAME is $FREE and LENGTH indicates the number of unused blocks in the logical disc, for example:


LD
user file
free blocks
$FREE entry
number indicated by LENGTH field in $FREE entry
subcatalog
entry describing user file

A user can connect his terminal to one logical disc at a time. This is done by means of the CONNECT command, which gives him read access, possibly together with other users, to the files in that logical disc.

If, however, the user correctly specifies the protection key of a logical disc in the CONNECT command, the logical disc is reserved for writing and he becomes its exclusive user. He may now CREATE, DELETE, RENAME, or write to files in that logical disc, and no other user may connect his terminal to it.

CONNECT and other commands related to the use of logical discs are described in the remaining sections of this chapter, while CREATE, DELETE, RENAME, and other statements related to the use of files are described in Chapter 8.

7.2 CONNECT

Format

CONNECT <ldname> [,<expr>]

<ldname>: a logical disc expressed as a string literal or by means of a variable.

<expr>: a numeric expression which evaluates to the protection key.

Use

As a command or statement to connect the user's terminal to a logical disc.

Remarks

1. If the user CONNECTs without specifying a protection key, he may only read from the logical disc <ldname>.

2. If the user specifies a protection key and the value is correct, he becomes the exclusive user of <ldname> and may now write to as well as read from <ldname>, whereas no other user may CONNECT his terminal to <ldname>.

3. If a CONNECT command is given from a terminal which is already connected to a logical disc, a RELEASE command (see Sect. 7.7) will be automatically executed.

Examples

```
* CONNECT "DISC1"
* CONNECT "SUBAREA",637
10 CONNECT LDNAME$,KEY
```

7.3 COPY

Format

```
COPY "<ldname>:<filename1>","<filename2>"
```

<ldname>: a logical disc.

<filename1>: a file in <ldname> to be copied.

<filename2>: a file in the logical disc to which the terminal is connected.

Use

As a command to copy a file from any logical disc to a file in the logical disc to which the user"s terminal is connected.

Remarks

1. A file can be copied to a logical disc only if the user correctly specified the protection key of that logical disc in the CONNECT command (see Sect. 7.2).

2. The COPY command copies the file <filename1> in the logical disc <ldname> (provided that <ldname> is not reserved for writing by another user) to a file, <filename2>, in the logical disc to which the terminal is connected.

3. If <filename2> does not exist, a file will be created with the name <filename2>.

Example

```
* COPY "LIB:PROG1","PROG2"
```

7.4 INIT

Format

INIT <device>

<device>: a device expressed as a string literal.

Use

As a command to initialize the main catalog in a device containing logical discs.

Remarks

1. The INIT command can only be given from the master terminal.

2. Users may CONNECT their terminals to logical discs in <device> as soon as the INIT command has been executed.

3. When the INIT command is executed, the UNIT ID is output on the master terminal. (For a description of the UNIT ID, see the NEW function in the separate publication RC BASIC System Logical Disc Formatting Program Operating Guide.)

```
Example                                  Comment

* INIT "$FD0"
FORMATTING EXAMPLE 77.02.09              ←UNIT ID.
```

7.5 LOCK

Format

LOCK <device>

<device>: a device expressed as a string literal.

Use

As a command to lock a device, when changing discs or closing down the system, so that no user can CONNECT his terminal to a logical disc in that device.

Remarks

1. The LOCK command can only be given from the master terminal.

2. When the LOCK command has been executed, no user can CONNECT his terminal to a logical disc in <device>. Connected users may continue to access <device> until they RELEASE their terminals (see Sect. 7.7).

3. When the LOCK command is executed, a 4-digit number is output on the master terminal indicating the number of users whose terminals are still connected to logical discs in <device>.

4. IMPORTANT: The device, i.e. the flexible or moving-head disc that contains the logical discs, must never be removed from its drive unit until it has been LOCKed and the number of users is 0.

Example	Comment
* LOCK "$FD0" 0001	Do not remove the disc from drive unit 0, as the number of device users is 1. Ask the user who is still connected to RELEASE his terminal and then LOCK the device once more before removing the disc.

7.6 LOOKUP

Format

```
LOOKUP ["$LPT"]
```

Use

As a command to return a listing of the files (names and attributes) in the logical disc, if any, to which the user's terminal is connected at the moment.

Remarks

1. The first column in the listing contains the name of each file.

2. The second column indicates whether the file is a sequential access file (S) or a random access file (R).

3. The third column gives the number of the last block written in the file (sequential access) or the number of records in the file (random access).

4. The fourth column gives the number of the last byte written in the last block (sequential access file) or the record length in bytes (random access file).

5. The fifth column indicates the number of blocks allocated to the file. The number of bytes per block will depend on the device that contains the logical disc. If the device is a flexible disc, there are 128 bytes per block. If the device is a moving-head disc file, there are 512 bytes per block.

6. The listing will always include the file $FREE, where the fifth column indicates the number of unused blocks in the logical disc.

7. If the LOOKUP "$LPT" form of the command is used, the listing will be output with headings on the line printer.

8. The listing will be interrupted, if the user presses the ESCape key.

Example

```
* LOOKUP
SORT.SR    S  00006  00077  00006
SORT.SV    S  00005  00080  00010
INDATA     S  00002  00040  00002
LAES.SV    S  00002  00008  00002
DATA.BN    S  00001  00061  00030
$FREE         65479  00000  00444
```

Comment

The length in used bytes of the file SORT.SV is (5-1) x 128 + 80.

7.7 RELEASE

Format

```
RELEASE
```

Use

As a command or statement to disconnect the user's terminal from the logical disc, if any, to which it is connected.

Remarks

1. If the terminal is not connected to a logical disc, the RELEASE command will have no effect.

2. If a CONNECT command (see Sect. 7.2) is given from a terminal which is already connected to a logical disc, a RELEASE command will be automatically executed.

3. If any of the files in the logical disc are open, the error message 0115: OPEN FILES ON LD will be output and the terminal will not be disconnected.

Example

```
* RELEASE
100 RELEASE
```

7.8 USERS

Format

```
USERS <device>
```

<device>: a device expressed as a string literal.

Use

As a command to return the number of users whose terminals are connected to any logical disc in a device at the moment.

Remarks

1. The USERS command is intended as an aid in closing down the system (see the LOCK command, Sect. 7.5).

2. If the number of device users is 0, <device> can be LOCKed and the disc containing <device> can be removed from its drive unit.

Example	Comment
* USERS "$FD0" 0001	See the LOCK command.

8 Files and Related Statements

8.1 Introduction

The present section describes the file concept itself, while statements related to the use of files are described in the remaining sections of this chapter. For the catalog structure in RC BASIC, i.e. logical discs, see the introduction to Chapter 7.

8.1.1 Disc files and devices

Several of the statements described in the following sections have <filename> as an argument. A filename may be the name of a disc file or the name of a device. In many respects, a disc file and a device are one and the same, and most of the explanations in this chapter apply to both. Devices, however, cannot be CREATEd, DELETEd, or RENAMEd. A device, moreover, is used either for input or for output, whereas a disc file can be used for both.

8.1.2 Standard devices and reserved names

The system devices which can be used in RC BASIC are listed below. The names in parentheses should be used when the device is referenced as a file.

Line printer ($LPT)

Paper tape punch ($PTP)

Paper tape reader ($PTR)

Card reader ($CDR or $MCDR)

Flexible disc ($FD0, $FD1)

The card reader has two names, as it can be used in two different ways.

The flexible disc is referenced directly when the INIT, LOCK, and USERS commands are executed (see Ch. 7); otherwise, discs are only referenced indirectly, by a disc filename.

8.1.3 Block sizes

An RC BASIC disc file comprises a number of consecutive blocks in a logical disc. Each disc file is described separately by an entry in the subcatalog of the logical disc (see Ch. 7). The size of the blocks in the file depends on the type of device used to contain the logical disc and, hence, the file. If the device is a flexible disc, the block size is 128 bytes. If the device is a moving-head disc file, the block size is 512 bytes. One byte (8 bits) corresponds to one character, so that a string containing 10 characters will occupy 10 bytes, whereas numeric data will occupy 4 bytes per item.

8.1.4 Filenames and file sizes

A disc file can be CREATEd with a name and a size. A filename may contain from 1 to 8 characters. All characters are legal, but the first character in a disc filename must not be a dollar sign ($). The size of a file is the number of blocks allocated to the file, expressed as a number greater than or equal to zero.

A disc file can also be DELETEd or RENAMEd.

8.1.5 How files are used

Files can be used for many purposes. The user can SAVE/LOAD/CHAIN/RUN or LIST/ENTER a program to or from a file (see Chs. 3 and 9).

A file can also be used for data. In order to read data from or write data to a file, the user must first OPEN the file. The data can be in binary or ASCII (character) format. When the file is OPENed, the user must specify one of the following modes:

Mode 0 for binary input from or binary output to a random access file (READ FILE or WRITE FILE statement).

Mode 1 for binary input from a sequential access file (READ FILE statement).

Mode 2 for binary output to a sequential access file, when data is to be appended to previously written data (WRITE FILE statement).

Mode 3 for binary output to a sequential access file (WRITE FILE statement).

Mode 4	for binary input (only) from a random access file (READ FILE statement).
Mode 9	for ASCII input from a sequential access file (INPUT FILE statement).
Mode 11	for ASCII output to a sequential access file (PRINT FILE statement).

When the user no longer needs to access a file, he should CLOSE it.

8.1.6 Random access files

The data in a random access file is organized in individual records, which can be accessed directly. If a random access file is to be used for both reading and writing, it must be OPENed in mode 0. This can be done only if the user correctly specified the protection key of the logical disc in the CONNECT command (see Ch. 7). If a random access file is to be used for reading only, it can be OPENed in mode 4. In this case, the user does not have to specify the protection key of the logical disc in the CONNECT command.

8.1.7 Sequential access files

The data in a sequential access file can only be accessed in a sequential manner. When a sequential access file is OPENed, the system positions to the beginning of the file and the data is read or written starting from there. If, however, a sequential access file is OPENed in mode 2, the system will position after the last item written to the file, so that data can be appended to previously written data. Sequential access files can be OPENed in mode 1, 2, 3, 9, or 11.

If a file is OPENed for reading (writing), it must be CLOSEd and OPENed again before it can be used for writing (reading).

8.1.8 Write protection

A disc file can be accessed only if the user has CONNECTed his terminal to the logical disc that contains the file (see Ch. 7). If a disc file is to be CREATEd, DELETEd, RENAMEd, or written to, the user must correctly specify the protection key of the logical disc in the CONNECT command.

The remaining sections of this chapter contain separate descriptions of the CREATE, DELETE, and RENAME statements,

statements related to file input/output, and the EOF(X) function.

For the use of these statements as keyboard commands, see Appendix C.

CONNECT and other commands related to the use of logical discs are described in Chapter 7.

8.2 CLOSE FILE

Format

CLOSE [FILE(<file>)]

<file>: a numeric expression which evaluates to a user file number that was previously associated with a filename in an OPEN FILE statement (see Sect. 8.11).

Use

As a statement or command to dissociate a filename and a user file number so that the file no longer can be referenced.

Remarks

1. The CLOSE FILE statement may be used to close a file so that it can be re-opened by an OPEN FILE statement with a new mode argument.

2. The CLOSE form of the statement closes all open files.

Examples

```
100 CLOSE FILE(1)
200 CLOSE FILE(X+3)
300 CLOSE
```

8.3 CREATE

Format

CREATE <filename>,<size>[,<recl>]

<filename>: the name (1 to 8 characters) of the disc file to be created, expressed as a string literal or by means of a variable.

<size>: a numeric expression specifying either the length of the file in blocks (sequential access file) or the number of records in the file (random access file).

<recl>: a numeric expression specifying the record length in bytes. <recl> should be specified if, and only if, the file is to be used as a random access file.

Use

As a statement or command to create a file in the logical disc to which the user's terminal is connected.

Remarks

1. A file can be created only if the user correctly specified the protection key of the logical disc in the CONNECT command (see Ch. 7).

2. For files created in a logical disc contained in a flexible disc, the block size is 128 bytes. For files created in a logical disc contained in a moving-head disc file, the block size is 512 bytes.

3. If the user does not know how large a file will be, he can specify <size> equal to 0 when he creates it. This will reserve the remainder of the logical disc to which the terminal is connected. The file can then be used for output, and when it is CLOSEd (see Sect. 8.2), the system will truncate it. <size> must be positive, if the file is a random access file.

4. No more than one file created with <size> equal to 0 can be used, unless the files in question have already been CLOSEd once.

5. If <recl> is specified, the file can (only) be used as a random access file. (See also the OPEN FILE, READ FILE, and WRITE FILE statements.) <recl> must be positive and less than or equal to the block size of the device (i.e. flexible disc or moving-head disc file) containing the logical disc in which the file is created.

6. A random access file will be organized such that each physical block in the device will contain an integral number of records.

Example 1	Comment (1)
120 LET NAME$="PROC1.SR"	
130 CREATE NAME$,17	
140 CREATE "PROC2.SR",0	CREATE used as a statement.

Example 2	Comment (2)
* CREATE "DATAFILE",15,52	CREATE used as a command.

8.4 DELETE

Format

```
DELETE <filename>
```

<filename>: the name of the disc file to be deleted, expressed as a string literal or by means of a variable.

Use

As a statement or command to delete a file in the logical disc to which the user's terminal is connected.

Remarks

1. A file can be deleted only if the user correctly specified the protection key of the logical disc in the CONNECT command (see Ch. 7).

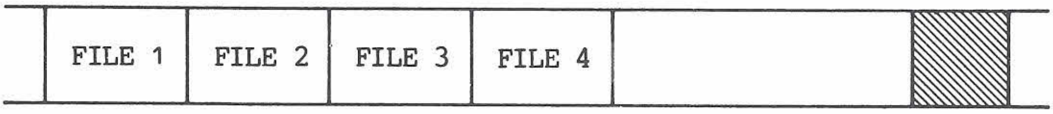
2. A file can be deleted only if all other files in the logical disc are closed.

3. Note: The deletion of a file may require some time.

   The figure below shows a logical disc containing four files. FILE 1 was created first and FILE 4 was created last.

   If, for example, FILE 2 is deleted, then FILE 3 and FILE 4 will be moved so that there will be no gaps in the logical disc.

   If the user wishes to delete more than one file, he should always delete starting with the file most recently created, as this will minimize the time required to move the files.



8.5 EOF(X) function

Format

EOF(<file>)

<file>: a numeric expression which evaluates to the number of a user file opened for reading (i.e. in mode 1 or 9).

Use

As a function to detect the end of data when transferring data from a file.

Remarks

1. The EOF(X) function returns an integer indicating whether or not the last READ FILE or INPUT FILE statement included an end of file delimiter.

2. If an end of file condition was detected, the function returns a value of +1; otherwise, a value of 0 is returned.

Example

See the examples under READ FILE (Sect. 8.14).

8.6 INPUT FILE

Format

```
                          {<var> }[ {<var> }]
INPUT FILE(<file>) [,]{<svar>}[,{<svar>}] ...
```

<file>: a numeric expression which evaluates to the number of a user file opened in mode 9.

<var, svar>: a list of one or more numeric or string variables which are assigned values read from a sequential access file.

Use

As a statement or command to read data in ASCII format from a sequential access file for the variables in the argument list.

Remarks

1. Each variable in the argument list must be of the same type (numeric or string) as the corresponding data item in the data file.

2. The data file must be formatted such that commas or carriage returns are used to separate numeric data items and quotation marks or carriage returns are used to separate string data items.

3. If the length of a string in the data file is greater than the length of the corresponding string variable in the argument list, the last part of the string will be skipped.

Example

```
40 OPEN FILE(1,9)"INFILE"
 .
 .
70 INPUT FILE(1)Z,Y,X,A$,B$
```

8.7 MAT INPUT FILE

Format

```
MAT INPUT FILE(<file>) [,] <mvar> [,<mvar>] ...
```

<file>: a numeric expression which evaluates to the number of a user file opened in mode 9.

<mvar>: a matrix variable.

Use

As a statement or command to read data in ASCII format from a sequential access file for the matrix variables in the argument list.

Remarks

1. The matrices must have been dimensioned before the statement is executed (see Ch. 6).

2. Values are assigned to the matrix elements by rows in ascending order.

3. The data file must be formatted such that commas or carriage returns are used to separate the data items.

8.8 MAT PRINT FILE

Format

```
MAT PRINT FILE(<file>) [,] <mvar> [,<mvar>] ...
```

<file>: a numeric expression which evaluates to the number of a user file opened in mode 11.

<mvar>: a matrix variable.

Use

As a statement or command to write matrix data in ASCII format to a sequential access file.

Remarks

See the MAT PRINT statement (Ch. 6).

8.9 MAT READ FILE

Format

```
MAT READ FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ...
```

<file>: a numeric expression which evaluates to the number of a user file opened in mode 0, 1, or 4.

<recno>: a numeric expression which evaluates to the number (> 0) of a record to be read from a random access file.

<mvar>: a matrix variable.

Use

As a statement or command to read data in binary format from a sequential access file or record of a random access file for the matrix variables in the argument list.

Remarks

1. The matrices must have been dimensioned before the statement is executed (see Ch. 6).

2. Values are assigned to the matrix elements by rows in ascending order.

3. If the attempt is made to read a record (from a random access file) which is longer than the record length specified for the file, the error message 0117: RECORD TOO LONG will appear.

8.10 MAT WRITE FILE

Format

```
MAT WRITE FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ...
       <file>:  a numeric expression which evaluates to the
                number of a user file opened in mode 0 or 3.

      <recno>:  a numeric expression which evaluates to the
                number (> 0) of a record to be written to a
                random access file.

       <mvar>:  a matrix variable.
```

Use

As a statement or command to write matrix data in binary format to a sequential access file or record of a random access file.

Remarks

1. The matrices must have been dimensioned before the statement is executed (see Ch. 6).

2. The values of the matrix elements are output by rows in ascending order.

3. If the attempt is made to write a record (to a random access file) which is longer than the record length specified for the file, the error message 0117: RECORD TOO LONG will appear.

8.11 OPEN FILE

Format

```
OPEN FILE(<file>,<mode>) [,] <filename>

          <file>:  a numeric expression which evaluates to a
                   number in the range 0 to 7 (the number of a
                   user file). This number is associated with
                   <filename> and used whenever the file is
                   referenced in other file input/output
                   statements.
```

<mode>: a numeric expression which evaluates to a number and specifies how the file is to be used (see Remarks).

<filename>: a disc file or a device expressed as a string literal or by means of a variable.

Use

As a statement or command to associate a disc file or a device with a user file number and to specify how the file is to be used.

Remarks

1. One of the following modes must be specified:

   Mode 0 for binary input from or binary output to a random access file (READ FILE or WRITE FILE statement).

   Mode 1 for binary input from a sequential access file (READ FILE statement).

   Mode 2 for binary output to a sequential access file, when data is to be appended to previously written data (WRITE FILE statement)

   Mode 3 for binary output to a sequential access file (WRITE FILE statement).

   Mode 4 for binary input (only) from a random access file (READ FILE statement).

   Mode 9 for ASCII input from a sequential access file (INPUT FILE statement).

   Mode 11 for ASCII output to a sequential access file (PRINT FILE or PRINT FILE USING statement).

2. A disc file must have been CREATEd (see Sect. 8.3) before it can be opened.

3. Random access files can only be opened in mode 0 or 4.

4. When a sequential access file is opened in mode 1, 3, 9, or 11, the system will position to the beginning of the file. When a sequential access file is opened in mode 2, the system

will position after the last item written to the file, so that data can be appended to previously written data.

Example 1	Comment (1)
10 NAME$="DATA1"	
20 OPEN FILE(0,0)NAME$	
30 OPEN FILE(1,11)"DATA2"	OPEN FILE used as a statement.

Example 2	Comment (2)
* OPEN FILE(6,9)"$PTR"	OPEN FILE used as a command.

8.12 PRINT FILE

Format

$$\text{PRINT FILE(<file>) [,]} \left\{\begin{array}{l}\text{<expr>}\\ \text{<slit>}\\ \text{<svar>}\end{array}\right\} \left[\left\{\begin{array}{l},\\ ;\end{array}\right\}\left\{\begin{array}{l}\text{<expr>}\\ \text{<slit>}\\ \text{<svar>}\end{array}\right\}\right] \ldots \left[\left\{\begin{array}{l},\\ ;\end{array}\right\}\right]$$

<file>: a numeric expression which evaluates to the number of a user file opened in mode 11.

<expr, slit, svar>: a list of one or more numeric or relational expressions, string literals, or string variables the values of which are written to a sequential access file.

Use

As a statement or command to write data in ASCII format to a sequential access file.

Remarks

1. The PRINT FILE statement is used to output data to an ASCII device, such as a line printer, or to a disc file for subsequent off-line printing.

2. Each item in the argument list must be separated from the next item by a comma or a semicolon. The argument list itself must be terminated by a carriage return.

3. Output formatting is the same as that described under the PRINT statement (see Ch. 3).

Example

```
10 OPEN FILE(2,11)"DATAFILE"
20 PRINT FILE(2)"RESULTS:"
30 PRINT FILE(2)X↑2,X↑3,X↑4
```

8.13 PRINT FILE USING

Format

$$\text{PRINT FILE(<file>) [,] USING <format>, }\left\{\begin{matrix}\text{<expr>}\\ \text{<slit>}\\ \text{<svar>}\end{matrix}\right\}\left[\left\{\begin{matrix},\\ ;\end{matrix}\right\}\left\{\begin{matrix}\text{<expr>}\\ \text{<slit>}\\ \text{<svar>}\end{matrix}\right\}\right]\ldots\left[\left\{\begin{matrix},\\ ;\end{matrix}\right\}\right]$$

<file>: a numeric expression which evaluates to the number of a user file opened in mode 11.

<format>: a string literal or string variable that specifies the format (see Remarks) for outputting the items in the argument list.

<expr>: a numeric or relational expression.

<slit>: a string literal.

<svar>: a string variable.

Use

As a statement to output the values of items in the argument list using a specified format.

Remarks

See the PRINT FILE statement (Sect. 8.12) and the PRINT USING statement (Ch. 3).

8.14 READ FILE

Format

$$\text{READ FILE(<file>[,<recno>]) [,]}\left\{\begin{matrix}\text{<var>}\\ \text{<svar>}\end{matrix}\right\}\left[\text{,}\left\{\begin{matrix}\text{<var>}\\ \text{<svar>}\end{matrix}\right\}\right]\text{ ...}$$

<file>: a numeric expression which evaluates to the number of a user file opened in mode 0, 1, or 4.

<recno>: a numeric expression which evaluates to the number (> 0) of a record to be read from a random access file.

<var, svar>: a list of one or more numeric or string variables which are assigned values read sequentially from a randomly accessed record or sequentially from a file.

Use

As a statement or command to read data in binary format from a sequential access file or record of a random access file for the variables in the argument list.

Remarks

1. Each variable in the argument list must be of the same type (numeric or string) as the corresponding data item in the data file.

2. One can, however, read string data items into numeric variables. (For each numeric variable, four bytes will be read.) This facility can be used to copy a file (see Ex. 2). Note: If the total number of bytes in the file is not divisible by 4, the error message 0107: END OF FILE may appear.

3. The EOF(X) function (see Sect. 8.5) can be used to detect an end of file condition in the file (see Examples).

4. If the attempt is made to read a record (from a random access file) which is longer than the record length specified for the file, the error message 0117: RECORD TOO LONG will appear.

Example 1

```
0010 OPEN FILE(0,1)"DATA"
0020 DIM TEXT$(25)
0030 READ FILE(0)TEXT$
0040 PRINT TEXT$
0050 READ FILE(0)A
0060 WHILE NOT EOF(0) DO
0070   PRINT A;
0080   READ FILE(0)A
0090 ENDWHILE
0100 CLOSE
```

```
THIS IS A DATA FILE
 1  2  3  4  5  6  7  8  9  10
```

Comment (1)

This program uses the file DATA, which is created in the program shown as an example of the WRITE FILE statement (see Sect. 8.16).

Example 2

```
0010 CREATE "DATA1",5
0020 OPEN FILE(0,1)"DATA"
0030 OPEN FILE(1,3)"DATA1"
0035 READ FILE(0)A
0040 WHILE NOT EOF(0) DO
0050   WRITE FILE(1)A
0060   READ FILE(0)A
0070 ENDWHILE
0080 CLOSE
```

Comment (2)

This program copies the file DATA to a new file, DATA1.

8.15 RENAME

Format

RENAME <filename1>,<filename2>

<filename1>: the name of the disc file to be renamed.

<filename2>: the new name of <filename1>.

Both arguments are expressed as string literals or by means of variables.

Use

As a statement or command to rename a file in the logical disc to which the user's terminal is connected.

Remarks

A file can be renamed only if the user correctly specified the protection key of the logical disc in the CONNECT command (see Ch. 7).

Example 1	Comment (1)
20 RENAME NAME$,"PROC3.SR"	RENAME used as a statement.
Example 2	Comment (2)
* RENAME "DATAFILE","FILE-3"	RENAME used as a command.

8.16 WRITE FILE

Format

```
                                ⎧<expr>⎫ ⎡ ⎧<expr>⎫⎤
WRITE FILE(<file>[,<recno>]) [,]⎨<slit>⎬ ⎢,⎨<slit>⎬⎥ ...
                                ⎩<svar>⎭ ⎣ ⎩<svar>⎭⎦
```

<file>: a numeric expression which evaluates to the number of a user file opened in mode 0 or 3.

<recno>: a numeric expression which evaluates to the number (> 0) of a record to be written to a random access file.

<expr, slit, svar>: a list of one or more numeric or relational expressions, string literals, or string variables the values of which are written sequentially to a randomly accessed record or sequentially to a file.

Use

As a statement or command to write data in binary format to a sequential access file or record of a random access file.

Remarks

1. A numeric data item is written as four bytes. A string data item is written as a number of bytes corresponding to the length of the string. The string is terminated by a NUL character (see App. D).

2. The result of a relational expression is written as one numeric item, the value of which is 1, if the expression is true (e.g. 3 > 0), or 0, if the expression is false (e.g. 0 > 3).

3. If the attempt is made to write a record (to a random access file) which is longer than the record length specified for the file, the error message 0117: RECORD TOO LONG will appear.

Example

```
0010 CREATE "DATA",5
0020 OPEN FILE(1,3)"DATA"
0030 WRITE FILE(1)"THIS IS A DATA FILE"
0040 FOR I=1 TO 10
0050   WRITE FILE(1)I
0060 NEXT I
0070 CLOSE FILE(1)
```

9 System Commands

9.1 Introduction

The statements and functions that are used for writing programs in RC BASIC are described in preceding chapters. RC BASIC, however, may also be used interactively to perform such functions as:

Maintenance of RC BASIC source programs
Desk calculator functions
Dynamic program debugging
File input/output

The present chapter describes commands for program development and execution.

Commands derived from RC BASIC statements, for desk calculator functions, program debugging, and file input/output, are described in Appendix C.

Commands used in conjunction with batch mode are described in Appendix B.

9.2 Command to delete program statements

Format

```
{<line n1>,<line n2>}
{<line n1>          }
{<line n1>,         }
{,<line n2>         }
```

<line n1>: the first statement to be deleted.

<line n2>: the last statement to be deleted.

Use

As a command to delete one or more statements in a program.

Remarks

The variations of the command have the following effects:

<line n1>,<line n2>	Deletes all lines with <line n1> <= line number <= <line n2>.
<line n1>	Deletes only the single line with line number = <line n1>.
<line n1>,	Deletes all lines with <line n1> <= line number.
,<line n2>	Deletes all lines with line number <= <line n2>.

Example

Comment

```
* LIST
0010 PRINT
0020 PRINT
0030 PRINT
0040 PRINT
0050 PRINT
0060 PRINT
0070 PRINT
0080 PRINT
0090 PRINT
0100 PRINT
```

```
* 20,40
* LIST
0010 PRINT
0050 PRINT
0060 PRINT
0070 PRINT
0080 PRINT
0090 PRINT
0100 PRINT
```

Lines 20-40 (= 20, 30, 40) are deleted.

```
* 60                                  Line 60 is deleted.
* LIST
0010 PRINT
0050 PRINT
0070 PRINT
0080 PRINT
0090 PRINT
0100 PRINT

* 90,                                 Lines 90-9999 (= 90, 100)
* LIST                                are deleted.
0010 PRINT
0050 PRINT
0070 PRINT
0080 PRINT

* ,70                                 Lines 1-70 (= 10, 50, 70)
* LIST                                are deleted.
0080 PRINT

*
```

9.3 AUTO

Format

```
     ⎡⎧<line n1>                          ⎫⎤
     ⎢⎪{STEP}                             ⎪⎥
AUTO ⎢⎨{,   }   <line n2>                 ⎬⎥
     ⎢⎪              {STEP}               ⎪⎥
     ⎣⎩<line n1>     {,   } <line n2>     ⎭⎦
```

<line n1>: the initial line number in a program.

<line n2>: the increment between line numbers in a program.

Use

As a command to provide automatic line numbers in a program, thereby making it easier to enter programs from a terminal.

Remarks

1. The terminal is released from AUTO mode by pressing the ESCape key.

2. AUTO can be used as a command for a file that contains statements; the statements can then be read into the current program storage area by means of the ENTER command (see Sect. 9.7).

3. The variations of the command have the following effects:

AUTO

Assigns numbers to a program starting with the default line number 0010 and with a default increment of 10 between line numbers.

AUTO <line n1>

Assigns numbers to a program starting with line number <line n1> and incrementing by <line n1> between line numbers.

AUTO $\left\{\begin{matrix}\text{STEP}\\ ,\end{matrix}\right\}$<line n2>

Assigns numbers to a program starting with the default line number 0010 and incrementing by <line n2> between line numbers.

AUTO <line n1>$\left\{\begin{matrix}\text{STEP}\\ ,\end{matrix}\right\}$<line n2>

Assigns numbers to a program starting with line number <line n1> and incrementing by <line n2> between line numbers.

```
Example                         Comment

* AUTO
0010 LET I=1
0020                            An empty line is ignored.
0020 LET TOOLONGNAME=5

ERR : 0011
NAME TOO LONG
0020 LET Y=X                    If an error occurs, the line is
0030                            repeated.
*
```

9.4 BATCH/BATCH "$LPT"

Format

```
BATCH ["$LPT"]
```

Use

As a command to place the terminal in batch mode and cause the system to start reading cards from the mark-sense card reader.

Remarks

1. Output from the jobs executed, i.e. listings, output from PRINT statements (see Ch. 3), and error messages, will appear on the terminal or, if the BATCH "$LPT" form of the command is used, on the line printer.

2. For a complete description of the batch mode of operation, see Appendix B.

9.5 BYE

Format

```
BYE
```

Use

As a command or statement to log the terminal off the system.

another part of the program, execution will then continue at the first statement following the IF-THEN statement.

2. If the value of <expr> is false (= 0), <statement> is not executed.

   Note: Since the internal representation of non-integer numbers may not be exact (.2 cannot be represented exactly, for example), it is advisable to test for a range of values when testing for a non-integer. If, for example, the result of a computation, A, was to be 1.0, a reliable test for 1 would be

   ```
   IF ABS(A-1.0)<1.0E-6 THEN ...
   ```

   If this test succeeded, A would be equal to 1 to within 1 part in 10↑6. This is approximately the accuracy of single-precision floating-point calculations.

Example 1

```
0010 LET I=10
0020 IF I>5 THEN GOTO 0040
0030 PRINT "DON'T ENTER HERE"
0040 PRINT "PRINT THIS"
0050 STOP

PRINT THIS
```

Comment (1)

The statement GOTO 40 is executed only if I > 5.

Example 2a

```
0010 LET A=5; B=5
0020 PRINT "A AND B ARE";
0030 IF A<>B THEN PRINT " NOT";
0040 PRINT " EQUAL"
0050 STOP

A AND B ARE EQUAL
```

Example

Comment

```
* LIST
0010 DEF FNF(X)=2↑X+2*X+2
0020 DATA 5,6,0
0030 PRINT " X ";"FNF(X)"
0040 READ X
0050 WHILE X<>0 DO
0060   PRINT X;FNF(X)
0070   READ X
0080 ENDWHILE
0090 PRINT "<13><10>SUPPLY NEW DATA (LINE 20)"
0100 STOP
0110 RESTORE 0020
0120 GOTO 0030


* RUN
 X FNF(X)
 5  44
 6  78

SUPPLY NEW DATA (LINE 20)

STOP
AT 0100
* 20 DATA 1,2,3,4,5,6,7,8,9,0
* CON
 X FNF(X)
 1  6
 2  10
 3  16
 4  26
 5  44
 6  78
 7  144
 8  274
 9  532

SUPPLY NEW DATA (LINE 20)

STOP
AT 0100
*
```

New data is supplied to the program before execution continues.

9.7 ENTER

Format

ENTER <filename>

<filename>: a disc file or a device expressed as a string literal or by means of a variable.

Use

As a command or statement to merge the statement lines from the disc file or the device specified by <filename> into the current program storage area.

Remarks

1. If an error is detected during the reading of a statement, the statement will be echoed on the terminal and an error message output (see App. A).

2. Only those statements in the current program that have line numbers equivalent to the line numbers of the ENTERed statements will be deleted. If, therefore, the current program (or a part of it) is not to be used, a NEW command (see Sect. 9.11) should be given prior to the ENTER command.

Example 1

```
* NEW
* ENTER "$PTR"
* ENTER "PROG.SR"
* LIST "$PTP"
```

Comment (1)

The user's program storage area is cleared. The program on paper tape and the program in file PROG.SR are merged. The resulting program is listed on paper tape.

Example 2

```
* LIST
0010 PRINT
0020 PRINT
0030 PRINT
0040 PRINT

* ENTER "$PTR"

* LIST
0010 PRINT I
0015 PRINT I
0020 PRINT
0025 PRINT I
0030 PRINT
0040 PRINT I
0050 PRINT I

*
```

Comment (2)

The current program and the program on paper tape are merged.

9.8 EOJ

Used only in batch mode. For description, see Appendix B.

9.9 LIST

Format

```
      ┌┌<line n1>                           ┐┐
      ││┌TO┐                                ││
LIST  ││└, ┘ <line n2>                      ││[<filename>]
      ││              ┌TO┐                  ││
      └└<line n1>     └, ┘  <line n2>       ┘┘
```

<line n1>: the first statement to be listed.

<line n2>: the last statement to be listed.

<filename>: a disc file or a device expressed as a string literal.

Remarks

1. The matrices must have been dimensioned before the statement is executed.

2. <mvar2> and <mvar3> must have the same dimensions.

3. The number of elements in <mvar2> (and <mvar3>) must not exceed the number of elements in <mvar1>.

4. The arithmetic is performed element by element with the resultant value assigned to the element in <mvar1>.

5. <mvar1> may appear on both sides of the equal sign.

6. After the addition or subtraction, <mvar1> will have the same dimensions as <mvar2> (and <mvar3>).

```
Example                                Comment

0010 DIM MATA(3,2),MATB(3,2)
0020 FOR I=1 TO 3
0030   FOR J=1 TO 2
0040     LET MATA(I,J)=I*10+J
0050     LET MATB(I,J)=2*(I*10+J)
0060   NEXT J
0070 NEXT I
0080 PRINT "MATA :  "
0090 MAT PRINT MATA
0100 PRINT "<13><10>MATB :  "
0110 MAT PRINT MATB
0120 MAT MATB=MATA+MATB                MATB appears on both sides
0130 PRINT "<13><10>MATA + MATB :"     of the equal sign.
0140 MAT PRINT MATB
```

terminal.

Examples	Comments
* LIST	The entire program will be listed on the terminal.
* LIST "$LPT"	The entire program will be listed on the line printer.
* LIST 100,500 "PROG1.SR"	Lines 100 through 500 will be listed to the file PROG1.SR.
* LIST 50	Line 50 will be listed on the terminal.

9.10 LOAD

Format

LOAD <filename>

<filename>: a disc file or a device expressed as a string literal.

Use

As a command to load a previously SAVEd program in binary format from the disc file or the device specified by <filename> into the user's program storage area.

Remarks

1. The LOAD command executes an implicit NEW command (see Sect. 9.11), thereby clearing any currently loaded program from core memory.

2. When a previously SAVEd program (see Sect. 9.16) has been LOADed, it can be LISTed (see Sect. 9.9), modified, or RUN (see Sect. 9.15).

Examples	Comments
* LOAD "$PTR"	<filename> is a device (paper tape reader).
* LOAD "PROG1.SV"	<filename> is a disc file.

9.11 NEW

Format

NEW

Use

As a command or statement to clear all currently stored program statements and variables from core memory and to close any open files (see Ch. 8).

Remarks

1. The user should clear his program storage area by means of a NEW command (or statement) before entering a new program so that statement lines from previous programs will not be executed along with the new program.

2. A NEW statement may appear as the last executable statement in a program, thereby clearing the program from core memory after program execution.

3. When used with an ON-ERR or ON-ESC statement (see Ch. 3), the NEW statement can be used to prevent unauthorized access to a program.

```
Example                                    Comment

* LIST
0010 LET NUMBER=5; I=0
0020 WHILE I<=NUMBER DO
0030   PRINT NUMBER;
0040   LET NUMBER=NUMBER+1; I=I+2
0050 ENDWHILE
0060 NEW                                   NEW used as a statement.

* RUN
 5  6  7  8  9  10
END
AT 0060
* LIST                                     This LIST command shows that
                                           the program has been cleared.
*
```

9.12 PAGE

Format

```
PAGE=<expr>
```

<expr>: a numeric expression in the range 0 <= <expr> <= 132.

Use

As a command or statement to set the right-hand margin of the terminal.

Remarks

1. The default page width (length of a print line) is 72 columns.

2. If the page width is set to zero, the system will regard the length of the print line as infinite and consequently not output an automatic carriage return and line feed in PRINT statements (see Ch. 3). The user may find this advantageous when using the X-Y addressing facilities of a video terminal.

Example   Comment

```
* LIST
0010 FOR I=1 TO 10
0020   PRINT I;
0030 NEXT I

* PAGE=30

* RUN
 1  2  3  4  5  6  7  8  9
 10
END
AT 0030
* PAGE=20

* RUN
 1  2  3  4  5  6
 7  8  9  10
END
AT 0030
*
```

The system outputs a carriage return and line feed when the page width (length of the print line) is exceeded.

9.13 PUNCH

Format

```
       ┌┌<line n1>                      ┐┐
       ││┌TO┐                           ││
PUNCH  │┤│, │<line n2>                  ├│
       ││           ┌TO┐                ││
       └└<line n1>  │, │<line n2>       ┘┘
```

<line n1>: the first statement to be punched.

<line n2>: the last statement to be punched.

Use

As a command to output part or all of the currently loaded program in ASCII to the terminal punch (when present).

Remarks

1. A PUNCHed listing is preceded by a leader and followed by a trailer, each containing 120 NUL characters (see App. D).

2. As the PUNCH command does not turn the terminal punch on and off, the following procedure is required:

   a. Type the desired PUNCH command, press the RETURN key, and immediately press the ON button on the punch.

   b. A NUL leader will be punched, followed by a listing of the desired lines of the current program, followed by a NUL trailer.

   c. When punching is completed, press the OFF button on the punch.

3. When part or all of a program is PUNCHed, a listing is output on the terminal simultaneously.

4. The variations of the command have the following effects:

PUNCH	Punches the entire program starting from the lowest numbered statement.
PUNCH <line n1>	Punches only the single statement at line number <line n1>.
PUNCH $\left\{\begin{matrix}\text{TO}\\ ,\end{matrix}\right\}$<line n2>	Punches from the lowest numbered statement through line number <line n2>.
PUNCH <line n1>$\left\{\begin{matrix}\text{TO}\\ ,\end{matrix}\right\}$<line n2>	Punches from line number <line n1> through line number <line n2>.

Example	Comment
* PUNCH 200 TO 500	Lines 200 through 500 will be punched.

9.14 RENUMBER

Format

$$\text{RENUMBER}\left[\left\{\begin{array}{l}\text{<line n1>}\\ \left\{\begin{array}{l}\text{STEP}\\ ,\end{array}\right\}\text{<line n2>}\\ \text{<line n1>}\left\{\begin{array}{l}\text{STEP}\\ ,\end{array}\right\}\text{<line n2>}\end{array}\right\}\right]$$

<line n1>: the initial line number in the current program.

<line n2>: the increment between line numbers in the current program.

Use

As a command to renumber the statements in the current program.

Remarks

1. The variations of the command have the following effects:

RENUMBER	Renumbers the current program starting with the default line number 0010 and with a default increment of 10 between line numbers.
RENUMBER <line n1>	Renumbers the current program starting with line number <line n1> and incrementing by <line n1> between line numbers.
$\text{RENUMBER}\left\{\begin{array}{l}\text{STEP}\\ ,\end{array}\right\}\text{<line n2>}$	Renumbers the current program starting with the default line number 0010 and incrementing by <line n2> between line numbers.
$\text{RENUMBER <line n1>}\left\{\begin{array}{l}\text{STEP}\\ ,\end{array}\right\}\text{<line n2>}$	Renumbers the current

program starting with line number <line n1> and incrementing by <line n2> between line numbers.

2. Line numbers are limited to four digits. If a RENUMBER command causes a line number to be greater than 9999, the command will be re-executed as:

```
RENUMBER 1 STEP 1
```

3. The RENUMBER command will also modify the line numbers in GOSUB, GOTO, ON-GOTO/GOSUB, and RESTORE statements (see Ch. 3) to agree with the new line numbers assigned to the current program.

4. References to non-existent lines are changed to 0000.

Example | Comment

```
* LIST
0001 LET NUMBER=5; I=0
0002 REPEAT
0004   PRINT NUMBER*I;
0007   LET I=I+1
0010 UNTIL I=NUMBER

* RENUMBER
```

The default values of <line n1>,<line n2> are 10,10.

```
* LIST
0010 LET NUMBER=5; I=0
0020 REPEAT
0030   PRINT NUMBER*I;
0040   LET I=I+1
0050 UNTIL I=NUMBER

* RENUMBER ,5

*LIST
0010 LET NUMBER=5; I=0
0015 REPEAT
0020   PRINT NUMBER*I;
0025   LET I=I+1
0030 UNTIL I=NUMBER

*
```

9.15 RUN/RUNL

Format

$$\left\{\begin{matrix}\text{RUN}\\ \text{RUNL}\end{matrix}\right\} \left[\left\{\begin{matrix}\langle\text{line no.}\rangle\\ \langle\text{filename}\rangle\end{matrix}\right\}\right]$$

<line no.>: the line number in the current program from which execution is to begin.

<filename>: a disc file or a device expressed as a string literal.

Use

As a command to execute the current program, either from the lowest numbered statement or from the line number specified by <line no.>, or to load and execute a previously SAVEd program as the current program.

Remarks

1. Output from PRINT statements (see Ch. 3) will appear on the terminal or, if the RUNL form of the command is used, on the line printer.

2. The variations of the command have the following effects:

RUN/RUNL

Clears all variables; undimensions all arrays and string variables; executes an implicit RESTORE command (see Ch. 3); resets the random number generator; runs the current program from the lowest numbered statement.

RUN/RUNL <line no.>

Retains all existing information, e.g. the values of variables and dimensioning, resulting from a previous execution of the current program; runs the current program from the line number specified by <line no.>.

This variation of the command allows program execution to be resumed retaining the current values of all variables and parameters. It may be used after the execution of a STOP statement (see Ch. 3) in the program, after the ESCape key has been pressed, or after an error has occurred, and will incorporate any changes made in the program after the program was stopped.

RUN/RUNL <filename>

LOADs a previously SAVEd program from the disc file or the device specified by <filename> (see Sect.

9.10), thereby clearing any currently loaded program from core memory; runs the previously SAVEd program from the lowest numbered statement.

Examples	Comments
* RUN	Runs the current program from the lowest numbered statement.
* RUNL 50	Runs the current program starting at line 50; output will appear on the line printer.
* RUN "$PTR"	Loads a program from paper tape and runs it from the lowest numbered statement.
* RUNL "PROG.SV"	Loads a program from the file PROG.SV and runs it from the lowest numbered statement; output will appear on the line printer.

9.16 SAVE

Format

SAVE <filename>

<filename>: a disc file or a device expressed as a string literal or by means of a variable.

Use

As a command or statement to write the currently loaded program, including the current values of all variables and parameters, in binary format to the disc file or the device specified by <filename>.

Remarks

1. If the program is written to a disc file, a new file named <filename> is created in the logical disc to which the terminal is connected (see Ch. 8). If <filename> already

exists, the program is written to this file. If an end of file condition is detected during the SAVEing operation, the error message 0107: END OF FILE will be output.

2. In the interests of conserving space on a SAVE device, one should add the statement

```
1 STOP
```

   to the program and RUN it before it is SAVEd. This will cause the core memory area which is used for variables during program execution to be truncated. The 1 STOP statement may then be deleted, before the program is SAVEd (see Ex. 2).

3. A SAVEd program can be LOADed, CHAINed, or RUN (see, respectively, Sect. 9.10, Ch. 3, and Sect. 9.15).

4. SAVEing, rather than LISTing (see Sect. 9.9), is a more efficient way to store large programs. The size of a program in binary format can be determined by means of the SIZE command (see Sect. 9.18). If the indicated size is less than the number of ASCII characters in the program, which may be ascertained by looking at the program, then SAVE should be used rather than LIST.

```
Example 1                           Comment (1)

* SAVE "$PTP"                       SAVE commands.
* SAVE "PROG1.SV"

200 SAVE "$PTP"                     SAVE statements.
200 SAVE "PROG1.SV"
```

```
Example 2                              Comment (2)

* LIST                                 Shows how one can save space
0010 DIM A(100)                        when SAVEing a program.
0020 FOR I=1 TO 100
0030   LET A(I)=I
0040 NEXT I

* SIZE
00422 BYTES USED                       ←Size before execution.
04578 BYTES LEFT

* RUN

END
AT 0040
* SIZE
00836 BYTES USED                       ←Size after execution.
04164 BYTES LEFT

* 1 STOP                               ←1 STOP statement inserted.
* RUN

STOP
AT 0001
* SIZE
00428 BYTES USED                       ←Size after second
04572 BYTES LEFT                       execution.

* 1                                    ←1 STOP statement deleted.
* SIZE
00422 BYTES USED
04578 BYTES LEFT

* SAVE "$PTP"                          ←Program SAVEd on paper
*                                      tape.
```

9.17 SCRATCH

Used only in batch mode. For description, see Appendix B.

9.18 SIZE

Format

```
SIZE
```

Use

As a command to return the number of bytes used by the current program and the numbers of bytes left.

Remarks

Even though no program is present, the SIZE command will indicate that approximately 350 bytes have been used, as these are always required to administer the execution of running programs.

Example

```
* SIZE
00836 BYTES USED
04164 BYTES LEFT
```

9.19 TAB

Format

```
TAB=<expr>
```

<expr>: a numeric expression in the range 1 <= <expr> <= page width specified by the PAGE command.

Use

As a command or statement to set the zone spacing between the print elements output by PRINT statements (see Ch. 3).

Remarks

1. The default zone spacing (width of a print zone) is 14 columns. This spacing allows five print zones per 72 character print line.

2. Since the maximum range of zone spacing depends on the PAGE command setting (see Sect. 9.12), it is wise to specify the page width (length of the print line) first and then specify the setting of the tabulation zones.

<u>Example</u>

```
* LIST
0010 FOR I=1 TO 10
0020   PRINT I,
0030 NEXT I

* PAGE=30

* TAB=10

* RUN
 1         2         3
 4         5         6
 7         8         9
 10
END
AT 0030
* TAB=5

* RUN
 1    2    3    4    5    6
 7    8    9    10
END
AT 0030
*
```

9.20 TIME

Used only in batch mode. For description, see Appendix B.

Examples

```
* CONNECT "DISC1"
* CONNECT "SUBAREA",637
10 CONNECT LDNAME$,KEY
```

7.3 COPY

Format

```
COPY "<ldname>:<filename1>","<filename2>"
```

<ldname>: a logical disc.

<filename1>: a file in <ldname> to be copied.

<filename2>: a file in the logical disc to which the terminal is connected.

Use

As a command to copy a file from any logical disc to a file in the logical disc to which the user"s terminal is connected.

Remarks

1. A file can be copied to a logical disc only if the user correctly specified the protection key of that logical disc in the CONNECT command (see Sect. 7.2).

2. The COPY command copies the file <filename1> in the logical disc <ldname> (provided that <ldname> is not reserved for writing by another user) to a file, <filename2>, in the logical disc to which the terminal is connected.

3. If <filename2> does not exist, a file will be created with the name <filename2>.

Example

```
* COPY "LIB:PROG1","PROG2"
```

7.4 INIT

Format

INIT <device>

<device>: a device expressed as a string literal.

Use

As a command to initialize the main catalog in a device containing logical discs.

Remarks

1. The INIT command can only be given from the master terminal.

2. Users may CONNECT their terminals to logical discs in <device> as soon as the INIT command has been executed.

3. When the INIT command is executed, the UNIT ID is output on the master terminal. (For a description of the UNIT ID, see the NEW function in the separate publication RC BASIC System Logical Disc Formatting Program Operating Guide.)

Example	Comment
* INIT "$FD0"	
FORMATTING EXAMPLE 77.02.09	←UNIT ID.

7.5 LOCK

Format

LOCK <device>

<device>: a device expressed as a string literal.

Use

As a command to lock a device, when changing discs or closing down the system, so that no user can CONNECT his terminal to a logical disc in that device.

A Error Messages

A.1 Introduction

The errors that can occur during use of the RC BASIC system fall into three categories.

1) Errors detected during program entry or command execution

If an error is detected when an RC BASIC statement is entered or when a command is executed, an error message will be output in the following form:

```
ERR : <xxxx>
<text>
```

<xxxx>: a decimal error code less than 0100.

<text>: a brief description of the error.

If an error is detected while a program is being read from a file, RC BASIC will output the erroneous statement followed by the above error message on the current output device.

2) Errors detected during program execution

If an error is detected during the execution of a program, an error message will be output in the following form:

```
AT <yyyy> :
ERR : <xxxx>
<text>
```

<yyyy>: the line number of the erroneous statement.

<xxxx>: a decimal error code less than 0100.

<text>: a brief description of the error.

3) Errors detected during file operations

If an error is detected during an input/output operation, an error message will be output in a form similar to that of the messages output for errors of the first or second category; however, the text IOERR will replace the text ERR and the decimal error code, <xxxx>, will be greater than or equal to 0100.

Error messages, i.e. messages for errors detected during program entry or command execution as well as those detected during program execution, are listed and explained in numerical order in Section A.2.

I/O error messages, i.e. messages for errors detected during file operations, are listed and explained in numerical order in Section A.3.

A.2 Error messages

0000: ILLEGAL COMBINATION OF RELATIONAL OPERATORS

The legal combinations are: <, <=, =<, =, >=, =>, >, and <>.

0001: CHARACTER UNKNOWN

A character not used in RC BASIC was input. This message may appear as the result of a transmission error.

0002: SYNTAX ERROR

The structure of an RC BASIC statement is incorrect.

0003: NO CORE

No core memory is available for additional terminals at the moment.

0004: ILLEGAL KEY

The protection key specified in a CONNECT command must be in the interval [0,65535].

0005: ILLEGAL STATEMENT NUMBER

A statement number must be in the inverval [1,9999]. An illegal statement number may occur as the result of an overflow when the AUTO command is used.

0006: TOO MANY NAMES

Too many names (variables and procedures) have been declared. The maximum number allowed is 93. The number of names can be reduced by means of the following commands:

* LIST <filename>
* NEW
* ENTER <filename>

0007: COMMAND NOT EXECUTABLE FROM DEVICE

This error will occur, for example, if an ENTER file contains a command or if the attempt was made to execute BYE (and certain other commands) in batch mode.

0008: ILLEGAL PAGE/TAB COMMAND

The following rule applies to the two commands PAGE=<expr1> and TAB=<expr2>:

0 <= <expr1> <= 132 and 1 <= <expr2> <= <expr1>

0009: LINE TOO LONG

A program line (when translated to internal form) is too long.

0010: TIME LIMIT EXCEEDED

The time limit specified for a batch job has been exceeded.

0011: NAME TOO LONG

A name (variable or procedure name) may not exceed a length of 8 characters.

0012: ILLEGAL COMMAND

Attempt to use an RC BASIC statement as a command, but the statement is meaningful only within the context of a program.

0013: LINE NUMBER DOES NOT EXIST

A line number that was referenced, e.g. by LIST or RUN, does not exist in the program.

0014: PROGRAM TOO LARGE

Available core memory is insufficient for the program (statements). Program size may be reduced by deleting REM statements and other nonessentials.

0015: NO MORE DATA FOR READ

The last element in the last DATA statement has been read, but the system attempted to execute READ once more.

0016: ARITHMETIC ERROR

Either division by 0 or overflow.

0017: UNDEFINED VARIABLE

Attempt to use a variable in an expression, but the variable has not been defined, e.g. by LET or DIM.

0018: GOSUB-RETURN DEPTH

Nested subroutines (GOSUB statements) or procedures (EXEC statements) may not exceed a depth of 7 levels.

0019: RETURN WITHOUT GOSUB

A RETURN or ENDPROC statement was encountered without the previous execution of a GOSUB or EXEC statement.

0020: FOR-NEXT DEPTH

Nested FOR-NEXT loops may not exceed a depth of 7 levels.

0021: FOR WITHOUT NEXT

Every FOR statement must have a corresponding NEXT statement.

0022: NEXT WITHOUT FOR

A NEXT statement was encountered without the previous execution of the corresponding FOR statement.

0023: NO MORE ROOM FOR VARIABLES

No more core memory is available to assign space for variables. Program size may be reduced by deleting REM statements and other nonessentials.

0024: ILLEGAL USE OF <

The character < may be used in a string literal only in connection with the indication of a decimal value to represent a non-printing or special character. Thus the ASCII character Carriage Return, for example, is represented as <13>. If the character < itself is to appear as a character in a text string, it must be represented as <60>.

0025: NOT IMPLEMENTED

This message will appear, for example, if the user attempts to execute a BATCH command and the system has no card reader.

0026: ONLY ALLOWED FROM MASTER TERMINAL

INIT, LOCK, and certain other commands can only be given from the master terminal.

0027: ILLEGAL FILE NUMBER

A user file number must be in the interval [0,7].

0028: ORIGINAL DIMENSIONING EXCEEDED

Attempt to dimension an array previously dimensioned to a smaller number of elements.

0029: EXPRESSION TOO COMPLEX

An expression contains too many parentheses.

0030: ILLEGAL FILE LENGTH

The specified length of a sequential access file in blocks or the specified number of records in a random access file must be greater than or equal to 0.

0031: SUBSCRIPT ERROR

A subscript of a dimensioned variable exceeds the upper bound of the dimension for the array or is less than 1.

0032: UNDEFINED FUNCTION

A user function, i.e. FNa(d), has not been defined in a DEF statement.

0033: Not used.

0034: ILLEGAL FUNCTION ARGUMENT

A function containing an illegal argument was called.

0035: FORMAT ERROR IN PRINT USING

The format of a PRINT USING statement is incorrect.

0036: PRINT ELEMENT TOO LONG

A print element contains more characters than the print line can hold. (The length of the print line can be set by means of the PAGE command.)

0037: DETERMINANT IS ZERO

A matrix cannot be inverted if its determinant is equal to zero.

0038: VARIABLE NOT DIMENSIONED

A variable which has not been declared was used in the form <name>(<subscript>).

0039: SAME MATRIX ON BOTH SIDES OF EQUAL SIGN

Certain matrix operations, such as multiplication, cannot be performed if the same matrix appears on both sides of the equal sign.

0040: DIMENSIONAL ERROR IN MATRIX OPERATION

a) In matrix addition or subtraction, the two matrices on the right-hand side of the equal sign must have the same dimensions.

b) In matrix multiplication, the rules set forth in Chapter 6, Section 5, must be respected.

c) The total number of elements in the matrix on the left-hand side of the equal sign must not be less than the number of elements in the matrix which is the result of the matrix operation on the right-hand side of the equal sign.

0041: MATRIX NOT SQUARE

Only square matrices can be inverted.

0042: FILE ALREADY OPEN

Attempt to open a user file (numbered from 0 to 7) which is already open.

0043: Not used.

0044: FILE NOT OPENED

Attempt to reference a user file which has not been opened.

0045: PROC WITHOUT ENDPROC

A procedure (which is introduced by a PROC statement) does not end with an ENDPROC statement.

0046: PROCEDURE DOES NOT EXIST

A procedure which was called by means of an EXEC statement does not exist in the program.

0047: Not used.

0048: NOT A SAVE FILE

A LOAD command was given specifying the name of a file that does not contain a SAVEd program.

0049: Not used.

0050: IF-ENDIF DEPTH

Nested IF-ENDIF constructions may not exceed a depth of 7 levels.

0051: ELSE WITHOUT IF

An ELSE statement was encountered without the previous execution of an IF statement.

0052: IF/ELSE WITHOUT ENDIF

An IF/ELSE statement must have a corresponding ENDIF statement.

0053: WHILE WITHOUT ENDWHILE

Every WHILE statement must have a corresponding ENDWHILE statement.

0054: WHILE-ENDWHILE DEPTH

Nested WHILE-ENDWHILE loops may not exceed a depth of 7 levels.

0055: ENDWHILE WITHOUT WHILE

An ENDWHILE statement was encountered without the previous execution of the corresponding WHILE statement.

0056: ENDIF WITHOUT IF

An ENDIF statement was encountered without the previous execution of an IF statement.

0057: REPEAT-UNTIL DEPTH

Nested REPEAT-UNTIL loops may not exceed a depth of 7 levels.

0058: UNTIL WITHOUT REPEAT

An UNTIL statement was encountered without the previous execution of the corresponding REPEAT statement.

0059: CASE WITHOUT WHEN, CASE ERROR

A CASE <expr> OF statement must either be matched by at least one WHEN <expr> statement or, if no match is found between the expression in the CASE statement and an expression in a WHEN statement, be followed immediately by one or more statements.

0060: CASE WITHOUT ENDCASE

A CASE statement must have a corresponding ENDCASE statement.

0061: ENDCASE WITHOUT CASE

An ENDCASE statement was encountered without the previous execution of a CASE statement.

0062: WHEN WITHOUT CASE

A WHEN statement was encountered without the previous execution of a CASE statement.

0063: CASE DEPTH

This message will appear when there is a programming error in the user's program, viz. a loop that causes CASE <expr> OF, but not ENDCASE, to be executed.

0064: NOT A DIMENSIONED VARIABLE

A simple variable has been used in the form <name> (<subscript>).

0065: ILLEGAL TYPE

An expression is of the wrong type.

0066: TYPE CONFLICT

A variable is not of the same type as that with which the user attempted to equate it.

Error codes 0067 through 0076 are all accompanied by the text SYSTEM ERROR. If a SYSTEM ERROR message appears, please fill out an RC Error Report, stating the error code.

Error codes 0077 through 0099 are not used.

A.3 I/O error messages

I/O errors, i.e. errors detected during file operations, are divided into two groups.

1) I/O error messages 0100 through 0119, 135, and 136

This group comprises errors recognized by RC BASIC.

2) I/O error messages 0120 through 0134

This group comprises errors recognized by the MUS or DOMUS operating system.

The two groups are described in Sections A.3.1 and A.3.2, respectively.

A.3.1 I/O error messages 0100 through 0119, 135, and 136

0100: FILE UNKNOWN

Attempt to reference a non-existent file.

0101: FILE OPENED INCORRECTLY

A file was opened in the wrong mode, e.g. the line printer cannot be opened in a read mode.

0102: FILE IN USE

Attempt to reference a file already in use.

0103: ILLEGAL FILENAME

A disc filename begins with the illegal character $.

0104: NOT CONNECTED TO LD

Attempt to reference a disc file, but the user's terminal was not connected to a logical disc.

0105: ILLEGAL COMMAND TO LD

Attempt to create, delete, rename, copy, or write to a disc file, but the user did not specify the protection key of the logical disc in the CONNECT command.

0106: ILLEGAL FILE OPERATION

Attempt to reference an unopened file.

0107: END OF FILE

Attempt to read or write outside a file.

0108: FILE TOO LONG

The number of free blocks in the logical disc is not sufficient for creation of a file of the specified size.

0109: FILE EXISTS

Attempt to create a file that already exists in the logical disc.

0110: LD UNKNOWN

The user attempted to connect his terminal to a non-existent logical disc.

0111: DEVICE UNKNOWN

Attempt to access a non-existent device.

0112: DEVICE INITIALIZED

The main catalog in the specified device has already been intialized.

0113: LD RESERVED

The user attempted to connect his terminal to a logical disc which another user has reserved for writing.

0114: WRONG KEY

The user specified the wrong protection key for a logical disc.

0115: OPEN FILES ON LD

The user attempted to release his terminal from a logical disc, but one or more files were open.

0116: LD RESERVED ON DEVICE

The main catalog in the device on the specified drive unit cannot be initialized because the device was not locked properly when the disc containing the device was last removed from the drive unit.

This device must now be reset using the special formatting program (see the separate publication RC BASIC System Logical Disc Formatting Program Operating Guide).

0117: RECORD TOO LONG

Attempt to read or write a record (from or to a random access file) which was longer than the record length specified for the file.

0118: NO MORE FILE DESCRIPTORS

Every system configuration has a fixed number of file descriptors, corresponding to the total number of disc files which can be open at one time.

One or more disc files must therefore be closed before the file in question can be opened.

0119: ILLEGAL RECORDNO

The number of a record to be read from or written to a random access file is larger than the total number of records in the file or less than 1.

0135: SYSTEM ERROR

Please fill out an RC Error Report, stating the error code.

0136: LD IN USE ON DEVICE

Attempt to initialize a device while a logical disc in the device was in use. The logical disc in question must be released before the device can be initialized.

A.3.2 I/O error messages 0120 through 0134

Messages for I/O errors 0120 through 0134 do not include a descriptive text, as the meanings of the error codes, which are explained below, depend on the device causing the error. None of the devices makes use of all of the error codes.

Line printer errors

0120: Unit disconnected.

0121: Unit off line.

0126: Illegal operation. Unit reserved by another process.

0128: Paper fault. For Charaband printer: Overwriting of a line has occurred more than 8 times.

0129: Unit not ready.

0130: Error in paper movement control character. For Charaband printer: Parity error during the loading or printing of a line.

0131: Paper low. End of paper.

0133: Driver process not loaded.

0134: Paper runaway.

8.10 MAT WRITE FILE

Format

```
MAT WRITE FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ...
       <file>:  a numeric expression which evaluates to the
                number of a user file opened in mode 0 or 3.

      <recno>:  a numeric expression which evaluates to the
                number (> 0) of a record to be written to a
                random access file.

       <mvar>:  a matrix variable.
```

Use

As a statement or command to write matrix data in binary format to a sequential access file or record of a random access file.

Remarks

1. The matrices must have been dimensioned before the statement is executed (see Ch. 6).

2. The values of the matrix elements are output by rows in ascending order.

3. If the attempt is made to write a record (to a random access file) which is longer than the record length specified for the file, the error message 0117: RECORD TOO LONG will appear.

8.11 OPEN FILE

Format

```
OPEN FILE(<file>,<mode>) [,] <filename>

          <file>:  a numeric expression which evaluates to a
                   number in the range 0 to 7 (the number of a
                   user file). This number is associated with
                   <filename> and used whenever the file is
                   referenced in other file input/output
                   statements.
```

Flexible disc drive errors

0120:	Unit disconnected.
0121:	Unit off line.
0123:	Address field parity error.
0124:	Disc write-protected.
0125:	Output: Disc write-protected.
0126:	Illegal operation. Unit reserved by another process.
0127:	End of file.
0128:	Block size error. Record format conflict.
0130:	Parity error.
0131:	End of medium.
0132:	Position error.
0133:	Driver process not loaded.
0134:	Time out.

Moving-head disc drive errors

0120:	Unit disconnected.
0121:	Unit off line.
0126:	Illegal operation. Unit reserved by another process.
0128:	Block size error. Record format conflict.
0129:	Data channel overrun.
0130:	Parity error.
0131:	End of medium.
0132:	Position error. Seek failure.

0133: Driver process not loaded.

0134: Time out.

Incremental plotter errors

0126: Illegal operation. Unit reserved by another process.

0128: Block size error. Record format conflict.

0133: Driver process not loaded.

0134: Time out.

Card reader punch errors

0121: Unit not ready: Off line, stopped, disconnected, or in error. (Check the indicators.)

0122: Feed error.

0126: Illegal operation. Unit reserved by another process.

0127: Secondary hopper empty.

0128: Block size error. Record format conflict.

0129: Data channel overrun.

0130: Parity error. Data error.

0131: Primary hopper empty.

0133: Driver process not loaded.

0134: Hardware trouble.

Magnetic tape unit errors

0121: Unit off line.

0122: Unit rewinding.

0123: Input: Byte limit conflict. Noise record.

0125: Output: Write ring not mounted.

0126: Illegal operation. Unit reserved by another process.

0127: Input: End of file mark.

0128: Block size error. Record format conflict.

0129: Data channel overrun.

0130: Parity error.

0131: Output: End of tape sensed.

0132: Position error.

0133: Driver process not loaded.

0134: Blank tape. Position error. Wrong density.

Cassette tape unit errors

0121: Unit off line or disconnected. Cassette released.

0122: Unit not ready. Rewinding or position to beginning of tape.

0126: Illegal operation. Unit reserved by another process.

0127: Input: End of file mark.

0128: Block size error. Record format conflict.

0129: Buffer overflow. Data late.

0130: Parity or block check error.

0131: Output: End of tape sensed.

0132: Position error.

0133: Driver process not loaded.

0134: Time out. Output: Write-enable plugs removed.

B Batch Mode and Programming on Mark-Sense Cards

B.1 Batch jobs

The batch mode of operation permits the user to enter and run one or more complete jobs from the mark-sense card reader (when present).

RC BASIC source programs, written on mark-sense cards, are placed in the reader; when the BATCH/BATCH "$LPT" command is given from a terminal, the system starts reading the cards.

The cards, the contents of which are interpreted exactly as if they had been entered from a terminal, can contain all of the RC BASIC statements and commands with few exceptions.

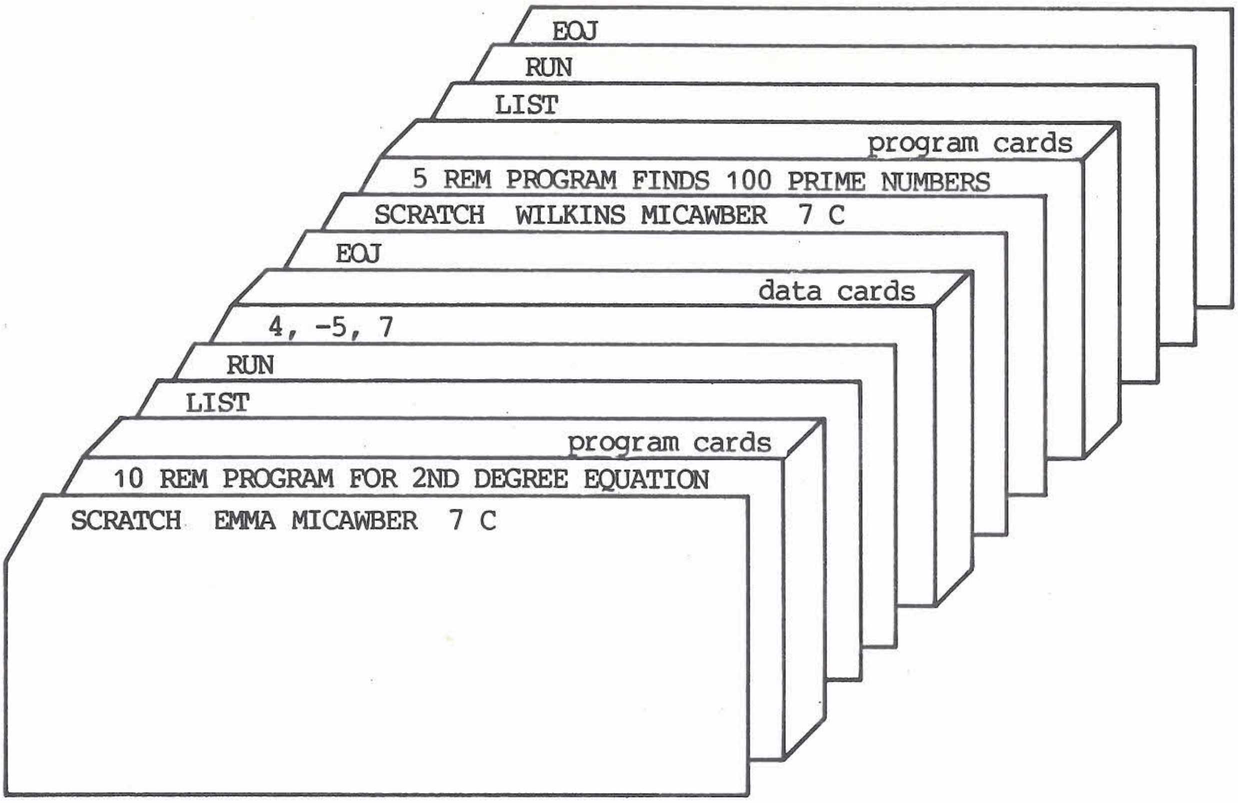
A stack of cards for batch entry is typically divided into several jobs.

Each job is initiated by a card containing a SCRATCH command (see Sect. B.5) and terminated by a card containing the EOJ command (see Sect. B.4).

Between the SCRATCH command and the EOJ command there can be an RC BASIC source program on cards, each of which contains one statement, and following the program there can be one or more cards containing one command each, e.g. LIST and RUN.

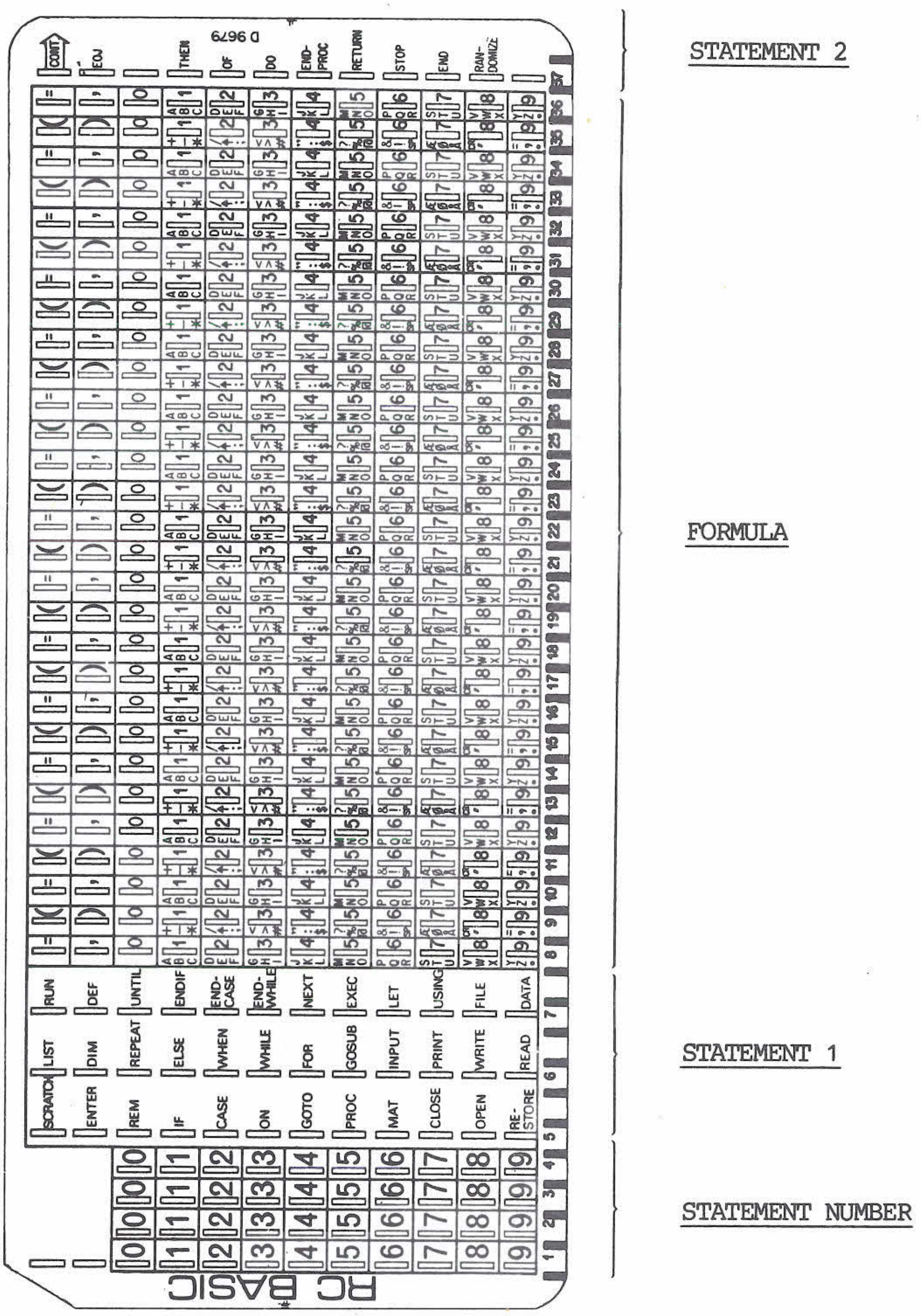
A RUN command may be followed by cards containing data for the program, which is read by means of INPUT statements (see Ch. 3) in the program. Such data cards are marked only in the FORMULA section of the card (see Sect. B.2).

The figure below shows a card stack containing two batch jobs.


EOJ
RUN
LIST
program cards
5 REM PROGRAM FINDS 100 PRIME NUMBERS
SCRATCH WILKINS MICAWBER 7 C
EOJ
data cards
4, -5, 7
RUN
LIST
program cards
10 REM PROGRAM FOR 2ND DEGREE EQUATION
SCRATCH EMMA MICAWBER 7 C

B.2 Mark-sense cards

The cards used for batch jobs are 37 column mark-sense programming cards. The RC BASIC mark-sense card looks like this:



The mark-sense card is not punched; instead, information is written on it simply by marking one or more fields with a soft pencil, e.g. No. 2.

As may be seen from the figure on the preceding page, the RC BASIC mark-sense card is divided into four sections, from left to right:

STATEMENT NUMBER	section (columns 1-4)
STATEMENT 1	section (columns 5-7)
FORMULA	section (columns 8-36)
STATEMENT 2	section (column 37)

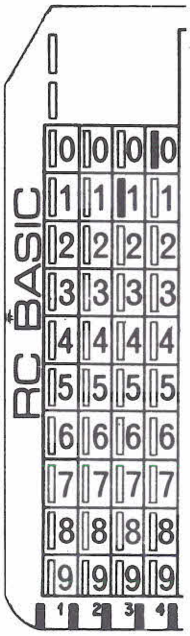
B.2.1 STATEMENT NUMBER

The STATEMENT NUMBER section (columns 1-4) is used for statement numbers.

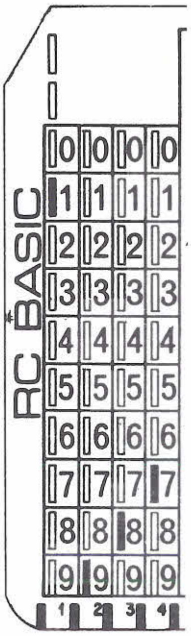
An RC BASIC program consists of statements, each of which begins with a statement number in the range 1 to 9999. A statement number is written by making at the most one mark in each of the columns 1 to 4.

Examples

10 :



1987:



B.2.2 STATEMENT 1

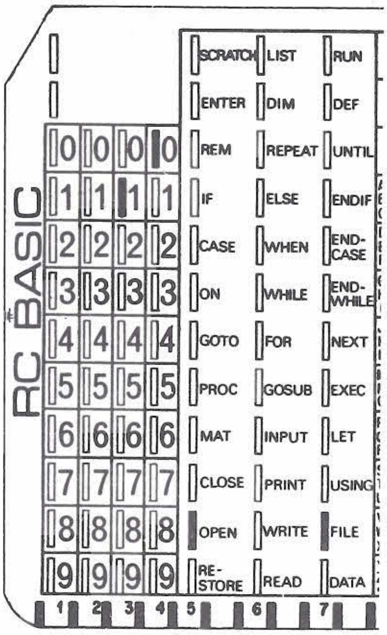
The STATEMENT 1 section (columns 5-7) permits the user to write one or more RC BASIC words simply by marking one field for each word.

None of the three columns in this section may contain more than one mark. Some of the RC BASIC words are commands, e.g. LIST or RUN, and may not be preceded by a line number. Other words, e.g. ENTER or CLOSE, may be used with or without a line number, i.e. either as statements or commands. Still other words can be used only with a line number, e.g. PROC or ENDPROC.

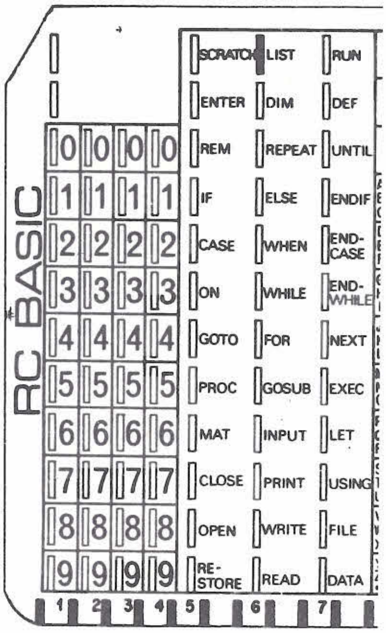
Whether a word must have, may have, or may not have a line number can be seen from the statement and command descriptions found in Chapters 3, 6, 7, 8, and 9.

Examples

10 OPEN FILE :



LIST :



B.2.3 STATEMENT 2

The STATEMENT 2 section (column 37) is used as follows:

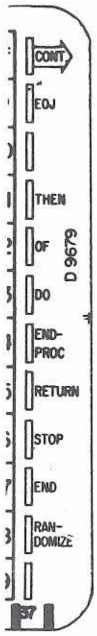
The CONT field should be marked whenever an RC BASIC statement fills more than one card. The system will then continue reading from the next card, skipping the STATEMENT NUMBER section of that card. A statement can theoretically be continued on any number of cards.

When the EOJ field is marked, the system will terminate the job. No other fields on the card should be marked.

The THEN, OF, and DO fields are used in conjunction with the words IF, CASE, and WHILE, respectively.

ENDPROC, RETURN, STOP, END, and RANDOMIZE are normal RC BASIC statements.

The STATEMENT 2 section looks like this:



B.2.4 FORMULA

The FORMULA section (columns 8-36) is used for that part of an RC BASIC statement which cannot be written in the STATEMENT 1 and STATEMENT 2 sections.

Each of the columns 8 to 36 contains twelve vertical fields. The first field from the top is field number 12, the second field from the top is field number 11, and the remaining fields are numbered 0 to 9.

B.2.4.1 Even numbered columns

Columns with even numbers (i.e. 8, 10, ..., 36) are used for the letters A to Z, the digits 0 to 9, and the following special characters:

=	(field 12)
,	(field 11)
.	(field 9)

B.2.4.2 Odd numbered columns

Columns with odd numbers (i.e. 9, 11, ..., 35) are used for the digits 0 to 9 and the following special characters:

(	(field 12)
)	(field 11)
+ - *	(field 1)
/ ↑ ;	(field 2)
< > #	(field 3)
" : $	(field 4)
? % @	(field 5)
& ! SP	(field 6)
Æ Ø Å	(field 7)
CR '	(field 8)
= , .	(field 9)

Note: SP is the space character. CR outputs positioning to the leftmost character position on the print line and a line feed (see App. D). Æ, Ø, and Å are letters of the Danish alphabet.

B.2.4.3 Writing characters. The following characters are (or may be) written by marking only one field:

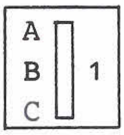
=	(field 12, even numbered columns)
(	(field 12, odd numbered columns)
,	(field 11, even numbered columns)
)	(field 11, odd numbered columns)
0 to 9	(fields 0 to 9, respectively)

All other characters are written by marking two fields:

The first field to be marked is the field in which the character itself appears on the card.

The second field to be marked is field 12, 11, or 0.

Which second field to mark is determined by the position of the character in the first field, for example:



The characters A, B, and C all appear in field 1.

Since the character A has the top position in field 1, it is written by marking field 1 and field 12.

Since the character B has the middle position in field 1, it is written by marking field 1 and field 11.

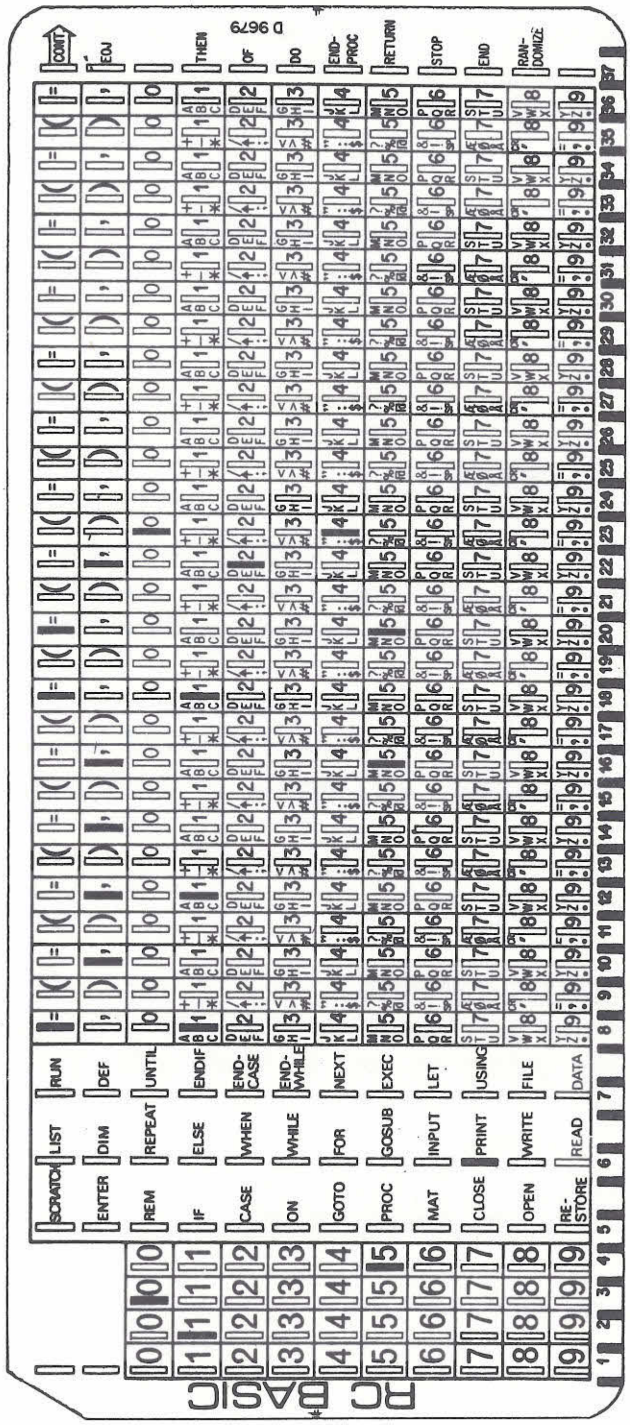
Since the character C has the bottom position in field 1, it is written by marking field 1 and field 0.

Example

The statement

```
105 PRINT A,B,NAME$
```

is written by marking a card as follows:



- $ (4 + 0)
- E (2 + 11)
- M (5 + 12)
- A (1 + 12)
- N (5 + 11)
- , (11)
- B (1 + 11)
- , (11)
- A (1 + 12)

- PRINT

- 5
- 0
- 1

Note

1. Blank spaces must be marked explicitly (the character SP). Columns having no marks whatsoever will be skipped by the card reader.

2. If a column contains more than the legal number of marks, the column will be skipped. (This can be utilized to skip a column containing an incorrectly marked field). Columns 1-7 may contain only one mark. Columns 8-36 may contain two marks. Column 37 may contain a mark in the CONT field and one other mark.

3. RC BASIC words not found in the STATEMENT sections can be written using the FORMULA section.

B.3 Batch mode

B.3.1 BATCH/BATCH "$LPT" command

A terminal can be placed in batch mode by giving the command BATCH or BATCH "$LPT". Before the command is given, the cards should be placed in the card reader and the reader should be ready, as the system will start reading cards at once.

If the BATCH form of the command is used, all output from the jobs executed, i.e. listings, output from PRINT statements (see Ch. 3), and error messages, will appear on the terminal.

If the BATCH "$LPT" form of the command is used, job output will be directed to the line printer.

B.3.2 Illegal statements and commands

As stated in Section B.1, there are a few RC BASIC statements and commands which cannot be used in batch mode. They are:

INIT

LOCK

USERS

AUTO

BATCH/BATCH "$LPT"

BYE

B.3.3 Time limit on jobs

A time limit can be placed on a job, so that when the job has run for a specified number of seconds, it will be interrupted.

The error message 0010: TIME LIMIT EXCEEDED will then be output and the next job started.

The next job is assumed to begin with a SCRATCH command.

When a job is started, the time limit is set by default to 60 seconds. This time limit can be changed by means of the TIME command (see Sect. B.6).

B.3.4 ESCape key

The ESCape key has a special function when the terminal is in batch mode.

When the ESCape key is pressed, the system will interrupt all current activity and output the following message on the terminal:

```
NEXT JOB(1), END OF BATCH(2), CONTINUE(3):
```

Here, the user should respond by typing one of the numbers (i.e. 1, 2, or 3) and pressing the RETURN key.

All according to the user's response, the following will now occur:

1	Cards will be read and skipped until a SCRATCH command is encountered or the reader is empty.
2	The terminal will be placed in interactive mode.
3	The next card will be read and its contents interpreted.

B.3.5 Return to interactive mode

When the BATCH/BATCH "$LPT" command has been given, the terminal will remain in batch mode until one of the following occurs:

1. The card reader becomes empty.

2. The user presses the ESCape key and then types the number 2.

3. An I/O error on the card reader is detected.

B.4 EOJ command

Format

EOJ

Use

As a command to terminate a job.

Remarks

1. The EOJ command executes an implicit NEW command (see Ch. 9).

2. Any logical disc that was connected by the job will be RELEASEd (see Ch. 7).

3. A card containing the EOJ command should always be the last card in a job.

B.5 SCRATCH command

Format

SCRATCH [<text>]

<text>: a text, i.e. a number of characters, which will be output as a heading.

Use

As a command to initiate a new job.

Remarks

1. If the command BATCH (see Sect. B.3) has been given, the

SCRATCH command will clear the display screen on the RC 822 or RC 823 terminal; if the command BATCH "$LPT" has been given, SCRATCH will cause the output of a form feed (see App. D) on the line printer.

2. If <text> is specified, <text> will be output as the first line displayed on the terminal or the first line on the new line printer page.

3. The job time limit is set to 60 seconds. This time limit can be changed by means of the TIME command (see Sect. B.6).

4. An implicit NEW command is executed (see Ch. 9).

5. Any logical disc that was connected by the previous job is RELEASEd (see Ch. 7).

6. A card containing a SCRATCH command should always be the first card in a job.

B.6 TIME command

Format

TIME=<val>

<val>: a numeric constant (expressing seconds).

Use

As a command to specify how many seconds a job may run.

Remarks

1. If the TIME command is not used, the system will interrupt the job in 60 seconds, starting from the execution of the SCRATCH command (see Sect. B.5).

2. The maximum specifiable time limit is 3600 seconds (1 hour).

3. The time allotted to a job is real time, i.e. the amount of central processing unit time which the job actually receives will depend on how many programs are being run at the same time from terminals in interactive mode.

C Other Interactive Uses of RC BASIC

C.1 Commands derived from RC BASIC statements

The interactive use of RC BASIC for source program maintenance is described in Chapter 9. This appendix describes other interactive uses of RC BASIC.

Many RC BASIC statements (see Chs. 3, 6, and 8) can be used as keyboard commands. When a statement is used as a command, it is entered without a preceding line number and terminated by pressing the RETURN key; the system then executes it immediately.

This facility is useful for:

Performing desk calculator functions, e.g. PRINT EXP(2.13)

Debugging programs dynamically

Performing file input/output, e.g. OPEN FILE(6,9) "$PTR"

Certain RC BASIC statements, which are meaningful only within the context of a program, cannot be used as keyboard commands. These statements are:

CASE-WHEN-ENDCASE

DATA

DEF

DELAY

END

EXEC

FOR-NEXT

GOSUB and RETURN

GOTO

ENDIF

ELSE

ON-ERR

ON-ESC

ON-GOTO/GOSUB

PRINT USING

PROC-ENDPROC

REM

REPEAT-UNTIL

STOP

WHILE-ENDWHILE

C.2 Desk calculator functions

Calculations can be performed using the PRINT command. A semicolon (;) may be used instead of the word PRINT.

The items in the argument list may be numeric or relational expressions, string literals, or string variables.

Values can be assigned to variables by means of the LET, INPUT, or READ command, or the current values of variables in a loaded program can be used.

```
Example                                Comments

* TAB=10
* K=SYS(14)/180                        K is the factor used when
* ;SIN(45*K),COS(45*K);TAN(45*K)       converting radians to
 .707107    .707107   1                degrees.
* ;SQR(169);SQR(27.45)
 13  5.23927
* ;"<12>"                              ← This PRINT command will
                                       clear the display screen on
*                                      the RC 822 or RC 823
                                       terminal.
```

C.3 Program debugging

The use of RC BASIC statements as keyboard commands permits programs to be debugged dynamically.

If, for example, a running program is producing the wrong output, it can be interrupted (ESCape key); the current values of the variables can then be examined (PRINT command) and changed (LET command) as required, before program execution is continued (see the CON or RUN <line no.> command, Ch. 9).

Example

Comments

```
0010 LET K=180/SYS(14)
0020 FOR I=0 TO 45 STEP 5
0030   PRINT USING "### #.####",I,SIN(K*I),
0040 NEXT I
```

```
* RUN
  0 0.0000        5 0.5597       10 0.9277
 15 0
STOP
AT 0030
```

The results indicate that the value of K is wrong.

```
* K=SYS(14)/180
* RUN 20
  0 0.0000        5 0.0872       10 0.1736
 15 0.2588       20 0.3420       25 0.4226
 30 0.5000       35 0.5736       40 0.6428
 45 0.7071
END
AT 0040
*
```

The user presses the ESCape key, assigns the right value to K, and resumes program execution from line 20.

C.4 File input/output

With the exception of PRINT FILE USING, the file input/output statements described in Chapter 8 can be used as keyboard commands to create a file, open a file, write data to a file, and so on.

If, for example, the error message 0044: FILE NOT OPENED is output, the user can open the file by means of the command

```
OPEN FILE(<file>,<mode>) <filename>
```

and program execution can then continue.

Use of the RELEASE command (see Ch. 7) may cause the error message 0115: OPEN FILES ON LD to appear, in which case the user should first give the command CLOSE and then RELEASE.

D ASCII Character Set

D.1 ASCII characters with their decimal and octal values

DEC	OCT	CHAR	DEC	OCT	CHAR	DEC	OCT	CHAR	DEC	OCT	CHAR
0	000	NUL	32	040	SP	64	100	@	96	140	`
1	001	SOH	33	041	!	65	101	A	97	141	a
2	002	STX	34	042	"	66	102	B	98	142	b
3	003	ETX	35	043	#	67	103	C	99	143	c
4	004	EOT	36	044	$	68	104	D	100	144	d
5	005	ENQ	37	045	%	69	105	E	101	145	e
6	006	ACK	38	046	&	70	106	F	102	146	f
7	007	BEL	39	047	'	71	107	G	103	147	g
8	010	BS	40	050	(	72	110	H	104	150	h
9	011	HT	41	051	)	73	111	I	105	151	i
10	012	LF	42	052	*	74	112	J	106	152	j
11	013	VT	43	053	+	75	113	K	107	153	k
12	014	FF	44	054	,	76	114	L	108	154	l
13	015	CR	45	055	-	77	115	M	109	155	m
14	016	SO	46	056	.	78	116	N	110	156	n
15	017	SI	47	057	/	79	117	O	111	157	o
16	020	DLE	48	060	0	80	120	P	112	160	p
17	021	DC1	49	061	1	81	121	Q	113	161	q
18	022	DC2	50	062	2	82	122	R	114	162	r
19	023	DC3	51	063	3	83	123	S	115	163	s
20	024	DC4	52	064	4	84	124	T	116	164	t
21	025	NAK	53	065	5	85	125	U	117	165	u
22	026	SYN	54	066	6	86	126	V	118	166	v
23	027	ETB	55	067	7	87	127	W	119	167	w
24	030	CAN	56	070	8	88	130	X	120	170	x
25	031	EM	57	071	9	89	131	Y	121	171	y
26	032	SUB	58	072	:	90	132	Z	122	172	z
27	033	ESC	59	073	;	91	133	[(Æ)	123	173	{ (æ)
28	034	FS	60	074	<	92	134	\ (Ø)	124	174	¦ (ø)
29	035	GS	61	075	=	93	135	] (Å)	125	175	} (å)
30	036	RS	62	076	>	94	136	↑ (^)	126	176	~
31	037	US	63	077	?	95	137	← (_)	127	177	DEL

Note: Decimal values 91 through 93 and 123 through 125 can be used for Danish or other national characters.

D.2 Output of non-printing characters

Non-printing characters, i.e. characters which do not appear as keys on the terminal keyboard, can be output by means of the following sequence:

```
PRINT "<x>"
```

where x is the decimal value of the character to be output.

Only a few of the non-printing characters are used ordinarily, e.g. to move the paper on the line printer or to position the cursor on a video terminal.

The non-printing characters that occur most frequently are:

12 FF Form Feed. On the line printer, feeds the form to the top of the next page and positions to the leftmost character position on the print line.

13 CR Carriage Return. On the terminal, positions to the leftmost character position on the print line.

10 LF Line Feed. On the terminal, feeds one line vertically.

Note: To output positioning to the leftmost character position on the print line and a line feed, the sequence PRINT "<13><10>" must be used.

7 BEL Bell. On the terminal, causes the bell to ring once.

The characters used to position the cursor on a video terminal will depend on the terminal in question and may be found in the operating guide for the terminal.

E Reserved Words

This appendix contains an alphabetical list of the reserved words in the RC BASIC language. These words have special meanings and may not be used as variables.

ABS
AND
ATN
AUTO

BATCH
BYE

CALL
CASE
CHAIN
CHR
CLOSE
CON
CONL
CONNECT
COPY
COS
CREATE

DATA
DEF
DELAY
DELETE
DET
DIM
DIV
DO

ELSE
END
ENDCASE
ENDIF
ENDPROC
ENDWHILE
ENTER
EOF
EOJ
ERR
ESC
EXEC
EXP

FALSE
FILE
FNA ... FNÅ
FOR

GOSUB
GOTO

IDN
IF
INIT
INPUT
INT
INV

LEN
LET
LIST
LOAD
LOCK
LOG
LOOKUP

MAT
MOD

NEW
NEXT
NOT

OF
ON
OPEN
OR
ORD

PAGE
PRINT
PROC
PUNCH

RANDOMIZE
READ
RELEASE
REM
RENAME
RENUMBER
REPEAT
RESTORE
RETURN
RND
RUN
RUNL

SAVE
SCRATCH
SGN
SIN
SIZE
SQR
STEP
STOP
SYS

TAB
TAN
THEN
TIME
TINPUT
TO
TRN
TRUE

UNTIL
USERS
USING

WHEN
WHILE
WRITE

ZER

Note: FNA ... FNÅ represents the 29 reserved words FNA, FNB, ..., FNZ, FNÆ, FNØ, and FNÅ.

F Summary of Statements, Commands, and Functions

F.1 RC BASIC statements (Chapter 3)

Format/Description   Section/Use

```
CASE <expr> OF                                              3.2
  [<statements-0>]
WHEN <expr> [,<expr>] ...
   <statements-1>
┌     .                      ┐
│     .                      │
│     .                      │
│WHEN <expr> [,<expr>] ...   │
└   <statements-n>           ┘
ENDCASE [<comment>]
```

The expression following CASE is evaluated and compared with the expressions following WHEN. If there is a match in the ith WHEN statement, statements-i is executed. If no match is found, statements-0 is executed. Control is then transferred to the first statement following ENDCASE. — STATEMENT

```
CHAIN <filename> [THEN GOTO <lineno.>]                      3.3
```

Runs the SAVEd program referred to by a filename when the statement is encountered in the user's program. When used as a command, CHAIN will LOAD, but not execute, the SAVEd program. — STATEMENT or COMMAND

```
     ┌<val> ┐ ┌┌,<val> ┐┐                                   3.4
DATA └<slit>┘ └└,<slit>┘┘...
```

Provides values to be read into variables appearing in READ or MAT READ statements. — STATEMENT

```
DEF FN<a>(<d>) = <expr>                                     3.5
```

Used with the function FNa(d) to define a user function. — STATEMENT

```
DELAY = <expr>                                                  3.6
```

Interrupts program execution for a specified number of seconds. STATEMENT

```
    ⎧<svar>(<m>)           ⎫ ⎡ ⎧<svar>(<m>)           ⎫⎤        3.7
DIM ⎨<array>(<m>)          ⎬ ⎢,⎨<array>(<m>)          ⎬⎥...
    ⎩<array>(<row>,<col>)  ⎭ ⎣ ⎩<array>(<row>,<col>)  ⎭⎦
```

Defines the size of string variables or numeric variable arrays. STATEMENT or COMMAND

```
END [<comment>]                                                 3.8
```

Terminates execution of the program. STATEMENT

```
EXEC <name>                                                     3.10
```

Executes a procedure defined by PROC-ENDPROC. STATEMENT

```
FOR <control var> = <expr1> TO <expr2> [STEP <expr3>]           3.11
  <statements>
NEXT <control var>
```

FOR begins a FOR-NEXT loop and defines the number of times a block of statements is to be executed. NEXT is the last statement in the loop and changes the value of the control variable. STATEMENT

```
GOSUB <lineno.>                                                 3.12
 .
 .
 .
<statements>
RETURN [<comment>]
```

GOSUB transfers control to the first statement of a subroutine. RETURN is the last statement in a subroutine and returns control to the first statement following the GOSUB statement that called the subroutine. STATEMENT

```
GOTO <lineno.>                                                  3.13
```

Transfers control unconditionally to a statement not in normal sequential order . STATEMENT

```
IF <expr> [THEN] <statement>                                   3.14
```

Executes a single statement depending on whether an expression is true or false.
STATEMENT or COMMAND

```
IF <expr> [THEN] [DO]                                          3.15
  <statements>
ENDIF [<comment>]
```

Executes a block of statements depending on whether an expression is true or false.
STATEMENT

```
IF <expr> [THEN] [DO]                                          3.16
  <statements-1>
ELSE [<comment>]
  <statements-2>
ENDIF [<comment>]
```

Executes statements-1 if an expression is true, otherwise statements-2.
STATEMENT

```
                  {<var> } [             {,<var> } ]           3.17
INPUT [<slit-0>,] {<svar>} [[,<slit-n>]  {,<svar>} ] ...
```

Assigns values entered from the user's terminal to numeric or string variables.
STATEMENT or COMMAND

```
      {<var> }          [  {<var> }          ]                 3.18
[LET] {<svar>} = <expr> [; {<svar>} = <expr> ] ...
```

Assigns the value of an expression to a variable.
STATEMENT or COMMAND

```
ON ERR THEN <statement>                                        3.20
```

Enables the programmer to take special action, if an error occurs during program execution.
STATEMENT

```
ON ESC THEN <statement>                                        3.21
```

Enables the programmer to take special action, if the ESCape key is pressed during program execution.
STATEMENT

$$\text{ON }\langle\text{expr}\rangle\text{ [THEN] }\begin{Bmatrix}\text{GOTO}\\ \text{GOSUB}\end{Bmatrix}\ \langle\text{lineno.}\rangle\ [,\langle\text{lineno.}\rangle]\ \ldots$$ 3.22

Transfers control to one of several lines in a program depending on the computed value of an expression when the statement is executed. STATEMENT

$$\begin{Bmatrix};\\ \text{PRINT}\end{Bmatrix}\left[\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\left[\begin{Bmatrix},\\ ;\end{Bmatrix}\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\right]\ \ldots\ \right]\left[\begin{Bmatrix},\\ ;\end{Bmatrix}\right]$$ 3.24

Prints specified items on the user's terminal. STATEMENT or COMMAND

$$\text{PRINT USING }\langle\text{format}\rangle,\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\left[\begin{Bmatrix},\\ ;\end{Bmatrix}\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\right]\ \ldots\ \left[\begin{Bmatrix},\\ ;\end{Bmatrix}\right]$$ 3.25

Outputs the values of items in the argument list using a specified format . STATEMENT

PROC <name>
  <statements>
ENDPROC [<comment>] 3.26

Defines a procedure. When the procedure is called by EXEC, control is transferred to the first statement following PROC. ENDPROC is the last statement in a procedure and returns control to the first statement following the EXEC statement that called the procedure. STATEMENT

RANDOMIZE 3.27

Causes the random number generator to start at a different point in the sequence of random numbers generated by the function RND(X). STATEMENT or COMMAND

$$\text{READ }\begin{Bmatrix}\langle\text{var}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\left[\begin{Bmatrix},\langle\text{var}\rangle\\ ,\langle\text{svar}\rangle\end{Bmatrix}\right]\ldots$$ 3.28

Reads in values from DATA statements and assigns the values to the variables listed STATEMENT or COMMAND in the statement.

REM [<comment>] 3.29

Inserts explanatory comments within a program. STATEMENT

REPEAT [<comment>] 3.30
  <statements>
UNTIL <expr>

Executes a block of statements repetitively until an expression is true. The block of statements is always executed at least once. STATEMENT

RESTORE [<lineno.>] 3.31

Resets the data element pointer to the beginning of the data list or to a particular DATA statement. STATEMENT or COMMAND

STOP [<comment>] 3.33

Terminates execution of the current program. STATEMENT

TAB(<expr>) 3.35

Used in PRINT statements to tabulate the printing position to the column number evaluated from an expression. FUNCTION

WHILE <expr> [THEN] DO 3.36
  <statements>
ENDWHILE [<comment>]

Executes a block of statements repetitively while an expression is true. If the expression is false the first time WHILE is encountered, the block of statements is not executed even once. STATEMENT

F.2 RC BASIC functions (Chapter 4)

ABS(<expr>) 4.2

Returns the absolute (positive) value of an expression. FUNCTION

ATN(<expr>) 4.3

Calculates the angle, in radians, whose tangent is an expression. FUNCTION

COS(<expr>) 4.4

Calculates the cosine of an angle which is expressed in radians. FUNCTION

EXP(<expr>) 4.5

Calculates the value of e (2.71828) to the power of an expression. FUNCTION

FN<a>(<d>) 4.6

A user function which is defined by DEF (see Sect. F.1) and returns a numeric value. FUNCTION

INT(<expr>) 4.7

Returns the value of the nearest integer not greater than an expression. FUNCTION

LOG(<expr>) 4.8

Calculates the natural logarithm of an expression. FUNCTION

RND(<expr>) 4.9

Produces a pseudo random number between 0 and 1. FUNCTION

SGN(<expr>) 4.10

Returns the algebraic sign of an expression. FUNCTION

SIN(<expr>) 4.11

Calculates the sine of an angle which is expressed in radians. FUNCTION

SQR(<expr>) 4.12

Computes the square root of an expression. FUNCTION

SYS(<expr>) 4.13

Returns system information, based on an expression which is evaluated to an integer, as follows: FUNCTION

0	Time of day.
1	Day.
2	Month.
3	Year.
4	Terminal port number.
5	Time used since terminal was logged on.
6	Number of file I/O statements executed.
7	Error code of last run-time error.
8	File number of last file referenced.
9	Page size.
10	Tab size.
11	Hour.
12	Minutes past last hour.
13	Seconds past last minute.
14	Constant π.
15	Constant e.

TAN(<expr>) 4.14

Calculates the tangent of an angle which is expressed in radians. FUNCTION

The following functions are described in other sections: the printing function TAB(X) in F.1; the string functions CHR(X), LEN(X$), and ORD(X$) in F.3; the matrix function DET(X) in F.4; and the function EOF(X) in F.6.

F.3 String functions (Chapter 5)

CHR(<expr>) 5.2

Returns the character corresponding to the number found as an expression modulo 128. FUNCTION

LEN($\begin{Bmatrix}\langle\text{svar}\rangle\\ \langle\text{slit}\rangle\end{Bmatrix}$) 5.3

Returns the current number of characters in a string. FUNCTION

ORD($\begin{Bmatrix}\langle\text{svar}\rangle\\ \langle\text{slit}\rangle\end{Bmatrix}$) 5.4

Returns the decimal number of the first character of a string. FUNCTION

F.4 Matrix statements (Chapter 6)

MAT <mvar1> = <mvar2> 6.3

Copies the elements of one matrix to another matrix. STATEMENT or COMMAND

MAT <mvar1> = <mvar2>$\begin{Bmatrix}+\\ -\end{Bmatrix}$<mvar 3> 6.4

Performs the scalar addition or subtraction of two matrices. STATEMENT or COMMAND

MAT <mvar1> = $\begin{Bmatrix}\langle\text{mvar2}\rangle\\ (\langle\text{expr}\rangle)\end{Bmatrix}$ * <mvar3> 6.5

Performs the multiplication of one matrix by another matrix or by a scalar. STATEMENT or COMMAND

<var> = DET(<expr>) 6.6

Returns the determinant of the last matrix inverted by a MAT INV statement. FUNCTION

MAT <mvar> = CON 6.7

Initializes a matrix such that all elements are set to one.

STATEMENT or COMMAND

MAT <mvar> = IDN 6.8

Initializes a matrix such that all elements (i,i) are set to one and the remaining elements are set to zero.

STATEMENT or COMMAND

MAT INPUT <mvar1> [,<mvar2>, ... ,<mvar-n>] 6.9

Assigns numeric values entered from the user"s terminal to the elements of one or more matrices.

STATEMENT or COMMAND

MAT <mvar1> = INV(<mvar2>) 6.10

Inverts a matrix and assigns the resultant element values to another matrix.

STATEMENT or COMMAND

MAT PRINT <mvar> $\left[\left\{\begin{matrix};\\,\end{matrix}\right\}\text{mvar>}\right]$ ... [;] 6.11

Outputs the values of the elements of one or more matrices on the user"s terminal.

STATEMENT or COMMAND

MAT READ <mvar> [,<mvar>] ... 6.12

Reads in numeric values from DATA statements and assigns the values to the elements of one or more matrices.

STATEMENT or COMMAND

MAT <mvar1> = TRN(<mvar2>) 6.13

Transposes a matrix and assigns the resultant element values to another matrix.

STATEMENT or COMMAND

MAT <mvar> = ZER 6.14

Initializes a matrix such that all elements are set to zero.

STATEMENT or COMMAND

F.5 Logical disc commands (Chapter 7)

CONNECT <ldname> [,<expr>] 7.2

Connects the user's terminal to a logical disc for reading or, if the protection key is correctly specified, for both reading and writing.

COMMAND or STATEMENT

COPY "<ldname>:<filename1>","<filename2>" 7.3

Copies a file (<filename1>) from any logical disc (<ldname>) to a file (<filename2>) in the logical disc to which the terminal is connected.

COMMAND

INIT <device> 7.4

Initializes the main catalog in a device containing logical discs.

COMMAND

LOCK <device> 7.5

Locks a device, when changing discs or closing down the system, so that no user can connect his terminal to a logical disc in that device.

COMMAND

LOOKUP ["$LPT"] 7.6

Returns a listing of the files in the logical disc to which the terminal is connected.

COMMAND

RELEASE 7.7

Disconnects the user's terminal from the logical disc to which it is connected.

COMMAND or STATEMENT

USERS <device> 7.8

Returns the number of users whose terminals are connected to any logical disc in a device.

COMMAND

F.6 File statements (Chapter 8)

CLOSE [FILE(<file>)] 8.2

Dissociates a filename and a user file number (see OPEN FILE) so that the file no longer can be referenced. The CLOSE form of the statement closes all open files. STATEMENT or COMMAND

CREATE <filename>,<size>[,<recl>] 8.3

Creates a file in the logical disc to which the user's terminal is connected. STATEMENT or COMMAND

DELETE <filename> 8.4

Deletes a file in the logical disc to which the user's terminal is connected. STATEMENT or COMMAND

EOF(<file>) 8.5

Returns a value of +1, if an end of file condition was detected in the last INPUT FILE or READ FILE statement; otherwise, a value of 0 is returned. <u>FUNCTION</u>

```
                        ⎧<var> ⎫⎡  ⎧<var> ⎫⎤
INPUT FILE(<file>) [,]  ⎩<svar>⎭⎣, ⎩<svar>⎭⎦ ...      8.6
```

Reads data in ASCII format from a sequential access file for the variables in the argument list. STATEMENT or COMMAND

MAT INPUT FILE(<file>) [,] <mvar> [,<mvar>] ... 8.7

Reads data in ASCII format from a sequential access file for the matrix variables in the argument list. STATEMENT or COMMAND

MAT PRINT FILE(<file>) [,] <mvar> [,<mvar>] ... 8.8

Writes matrix data in ASCII format to a sequential access file. STATEMENT or COMMAND

MAT READ FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ... 8.9

Reads data in binary format from a sequential access file or record of a random access file for the matrix variables in the argument list.

STATEMENT or COMMAND

MAT WRITE FILE(<file>[,<recno>]) [,] <mvar> [,<mvar>] ... 8.10

Writes matrix data in binary format to a sequential access file or record of a random access file.

STATEMENT or COMMAND

OPEN FILE(<file>,<mode>) [,] <filename> 8.11

Associates a filename, i.e. a disc file or a device, with a user file number so that the file can be referenced in other file I/O statements; also specifies how the file is to be used.

STATEMENT or COMMAND

$$\text{PRINT FILE}(\langle\text{file}\rangle)\ [,]\ \begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\left[\begin{Bmatrix},\\ ;\end{Bmatrix}\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\right]\ \ldots\ \left[\begin{Bmatrix},\\ ;\end{Bmatrix}\right]$$

8.12

Writes data in ASCII format to a sequential access file.

STATEMENT or COMMAND

$$\text{PRINT FILE}(\langle\text{file}\rangle)\ [,]\ \text{USING}\ \langle\text{format}\rangle,\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\left[\begin{Bmatrix},\\ ;\end{Bmatrix}\begin{Bmatrix}\langle\text{expr}\rangle\\ \langle\text{slit}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\right]\ \ldots\ \left[\begin{Bmatrix},\\ ;\end{Bmatrix}\right]$$

8.13

Writes data in ASCII format to a sequential access file, using a specified output format.

STATEMENT

$$\text{READ FILE}(\langle\text{file}\rangle[,\langle\text{recno}\rangle])\ [,]\ \begin{Bmatrix}\langle\text{var}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\left[,\begin{Bmatrix}\langle\text{var}\rangle\\ \langle\text{svar}\rangle\end{Bmatrix}\right]\ \ldots$$

8.14

Reads data in binary format from a sequential access file or record of a random access file for the variables in the argument list.

STATEMENT or COMMAND

```
RENAME <filename1>,<filename2>                                      8.15
```

Renames a file in the logical disc to which the user's terminal is connected. — STATEMENT or COMMAND

```
                                ⎧<expr>⎫ ⎡  ⎧<expr>⎫⎤
WRITE FILE(<file>[,<recno>]) [,]⎨<slit>⎬ ⎢ ,⎨<slit>⎬⎥ ...           8.16
                                ⎩<svar>⎭ ⎣  ⎩<svar>⎭⎦
```

Writes data in binary format to a sequential access file or record of a random access file. — STATEMENT or COMMAND

F.7 System commands (Chapter 9)

```
⎧<line n1>,<line n2>⎫                                               9.2
⎪<line n1>          ⎪
⎨<line n1>,         ⎬
⎩,<line n2>         ⎭
```

Deletes one or more statements in a program. — COMMAND

```
     ⎡⎧<line n1>                         ⎫⎤                         9.3
     ⎢⎪⎧STEP⎫                            ⎪⎥
AUTO ⎢⎨⎩,   ⎭<line n2>                   ⎬⎥
     ⎢⎪            ⎧STEP⎫                ⎪⎥
     ⎣⎩<line n1>   ⎩,   ⎭<line n2>       ⎭⎦
```

Provides automatic line numbers in a program, thereby making it easier to enter programs from a terminal. — COMMAND

```
BATCH ["$LPT"]                                                      9.4
```

Places the terminal in batch mode and causes the system to start reading cards from the mark-sense card reader. Job output will appear on the terminal or, if the BATCH "$LPT" form of the command is used, on the line printer. — COMMAND

BYE 9.5

Logs the terminal off the system.

COMMAND or STATEMENT

$$\begin{Bmatrix}\text{CON}\\ \text{CONL}\end{Bmatrix}$$

9.6

Continues execution of the current program after the execution of a STOP statement in the program, after the ESCape key has been pressed, or after an error has occurred. Output from PRINT statements will appear on the terminal or, if the CONL form of the command is used, on the line printer.

COMMAND

ENTER <filename> 9.7

Merges the statement lines from the disc file or the device specified by a filename into the current program storage area.

COMMAND or STATEMENT

$$\text{LIST}\left[\begin{Bmatrix}\text{<line n1>}\\ \begin{Bmatrix}\text{TO}\\ ,\end{Bmatrix}\text{<line n2>}\\ \text{<line n1>}\begin{Bmatrix}\text{TO}\\ ,\end{Bmatrix}\text{<line n2>}\end{Bmatrix}\right]\ \text{[<filename>]}$$

9.9

Outputs part or all of the currently loaded program in ASCII format to the disc file or the device specified by a filename or, if no filename is specified, to the terminal.

COMMAND

LOAD <filename> 9.10

Loads a previously SAVEd program in binary format from the disc file or the device specified by a filename into the user's program storage area.

COMMAND

NEW 9.11

Clears all currently stored program statements and variables from core memory and closes any open files.

COMMAND or STATEMENT

PAGE=<expr> 9.12

Sets the right-hand margin of the terminal.

COMMAND or STATEMENT

```
        ┌┌<line n1>              ┐┐                9.13
        ││┌TO┐                   ││
PUNCH   │{│, │<line n2>          }│
        ││            ┌TO┐       ││
        └└<line n1>   │, │<line n2>┘┘
```

Outputs part or all of the currently loaded program in ASCII format to the terminal punch (when present).

COMMAND

```
           ┌┌<line n1>                    ┐┐          9.14
           ││┌STEP┐                       ││
RENUMBER   │{│,   │<line n2>              }│
           ││            ┌STEP┐           ││
           └└<line n1>   │,   │<line n2>  ┘┘
```

Renumbers the statements in the current program.

COMMAND

```
┌RUN ┐  ┌┌<line no.> ┐┐                                 9.15
└RUNL┘  └└<filename> ┘┘
```

Executes the current program, either from the lowest numbered statement or from a specified line number, or loads and executes a previously SAVEd program as the current program. Output from PRINT statements will appear on the terminal or, if the RUNL form of the command is used, on the line printer.

COMMAND

SAVE <filename> 9.16

Writes the currently loaded program, including the current values of all variables and parameters, in binary format to the disc file or the device specified by a filename.

COMMAND or STATEMENT

SIZE 9.18

Returns the number of bytes used by the current program and the numbers of bytes left.

COMMAND

TAB=<expr> 9.19

Sets the zone spacing between the print elements output by PRINT statements. COMMAND or STATEMENT

F.8 Batch mode commands (Appendix B)

EOJ B.4

Terminates a job. COMMAND

SCRATCH [<text>] B.5

Initiates a job. COMMAND

TIME=<val> B.6

Specifies how many seconds a job may run. COMMAND

G Advanced Programming Examples

In course of preparation.

H Calling an Assembly Language Subroutine from RC BASIC

In course of preparation.

Index

Example 1

```
* LIST
0010 LET I=0
0020 REPEAT
0030   PRINT RND(I);
0040   LET I=I+1
0050 UNTIL I=4
0060 STOP
0070 GOTO 0010
```

```
* RUN
 .21132   .14464   .852625   .927054
STOP
AT 0060
* RUN
 .21132   .14464   .852625   .927054
STOP
AT 0060
* CON
 .162866   .433095   .563933   .20965
STOP
AT 0060
*
```

Comment (1)

The RUN command resets the sequence of random numbers; the CON command does not.

Example 2

```
* LIST
0010 LET I=0
0020 WHILE I<4 DO
0030   PRINT INT(25*RND(I));
0040   LET I=I+1
0050 ENDWHILE
```

```
* RUN
 5  3  21  23
END
AT 0050
*
```

Comment (2)

The program produces random integers in the range 0 to 24.

READER'S COMMENTS

RC BASIC
RCSL 42 - i 0673

A/S Regnecentralen maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback - your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

Do you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position:

Name:

Organization:

Address:

Department:

Date:

RETURN LETTER - CONTENTS AND LAYOUT

- - - - - - - - - - - - - - - - - - Fold here - - - - - - - - - - - - - - - - - -

- - - - - - - - - - Do not tear - Fold here and staple - - - - - - - - - - -

Affix
postage
here

A/S REGNECENTRALEN
Marketing Department
Falkoner Allé 1
DK-2000 Copenhagen F
Denmark

INTERNATIONAL

EASTERN EUROPE

A/S REGNECENTRALEN
Glostrup, Denmark, (02) 96 53 66

SUBSIDIARIES

AUSTRIA

RC – SCANIPS COMPUTER
HANDELSGESELLSCHAFT mbH
Vienna, (0222) 36 21 41

FINLAND

OY RC – SCANIPS AB
Helsinki, (90) 31 64 00

FRANCE

RC – COMPUTER S.a.r.l.
Paris, (1) 677 27 91

HOLLAND

REGNECENTRALEN (NEDERLAND) B.V.
Rotterdam, (010) 21 62 44

NORWAY

A/S RC – SCANIPS
Oslo, (02) 35 75 80

SWEDEN

RC – SCANIPS AB
Stockholm, (08) 34 91 55

SWITZERLAND

RC – SCANIPS (SCHWEIZ) AG
Basel, (061) 22 90 71

UNITED KINGDOM

REGNECENTRALEN LTD.
London, (01) 439 93 46

WEST GERMANY

RC – COMPUTER G.m.b.H.
Hannover, (0511) 63 99 51

REPRESENTATIVES

HUNGARY

HUNGAGENT AG
Budapest, 88 61 80

KUWAIT

KUWAIT – DANISH COMPUTER CO. S.A.K.
Kuwait, 51 05 10

CZECHOSLOVAKIA

KSNP KANCELARSKE STROJE N.P.
Praha, 27 00 01

TECHNICAL ADVISORY REPRESENTATIVES

POLAND

ZETO
Wroclaw, 44 54 31

RUMANIA

I.I.R.U.C.
Bucharest, 33 21 57

HUNGARY

NOTO–OSZV
Budapest, 66 84 11

REGNECENTRALEN
Scanips
COMPUTER

HEADQUARTERS: FALKONER ALLE 1 · DK-2000 COPENHAGEN F · DENMARK
PHONE: (01) 10 53 66 · TELEX: 16282 rc hq dk · CABLES: REGNECENTRALEN