Tiny BASIC

for

TinyBasRV

Software ver. 1.0

The TinyBasRV microcomputer contains a Tiny BASIC language interpreter with some limitations and, conversely, with a number of improvements. The interpreter provides the user interface (communication with the user) and also execution of the stored program.
After turning on the microcomputer, the message "Tiny BASIC x.x" appears on the VGA display indicating the current version of the interpreter. This document should have the same version as reported by the computer after starting. The microcomputer simultaneously tests the available hardware and possibly reports that the DISK f and the PCF8575 expander on the I2C bus are not available and that some functions are limited. To connect the "f" disk or expander, follow the instructions "What to do when..." at the end of this manual.

When interacting with the user, the microcomputer expects a command to be entered at the prompt in two possible formats:

STATEMENT (; STATEMENT) (; STATEMENT)

or

NUM STATEMENT (; STATEMENT) (; STATEMENT)

where STATEMENT represents a Tiny BASIC command that can be executed. If a line begins with a statement, it is executed immediately. If a line begins with a number NUM (in the range 0 to 127), the statement is not executed, but is stored in memory and becomes one of the lines of the program waiting to be executed.

Example:

PRINT 5+46

It immediately prints the result on the line 51
On the contrary

26 PRINT 5+46

does nothing. This line is stored at line number 26 in the program memory in the current "file" (see the FILE command). It remains inactively written in memory until the program is run. Only after the program is run is the command executed at the moment when line number 26 is processed.

There can be one, two, or three semicolon-separated statements on a line, which are executed sequentially from the left. Some statements, such as REM, GOTO, IF, cause the next statement (or statements) on the line to not be executed.

Writing a program

The program or interpreter commands are entered using the keyboard on the line at the blinking cursor. Printable characters appear on the screen immediately after entry. If you need to enter more than one "space" character, you can use the Tab key, which expands to 4 spaces. Any errors can be undone with the "Back Space" key. Moving on the line using the arrows is not possible, you can only delete characters one by one.
You can delete a misspelled line by entering a blank line. At the beginning of the line, type only the number of the deleted line and then press Enter. The misspelled line is corrected automatically after you type a new correct version of the line.
Tiny BASIC language commands are not written out character by character, but are entered by pressing 2 keys simultaneously. Commands are entered by pressing one key while simultaneously holding down the "Left CTRL" key. The list of allowed commands and their corresponding keys is in the summary table on the last page of this document.
Commands (STATEMENT) can be entered up to 3 at the same line. Most commands expect one or

more parameters in a specific, precisely defined format after the command itself. The list of parameters and their meaning is given below in the description of individual commands and also in the table on the last page. The main parameters of the commands are:

STRING
is a text string enclosed in quotation marks. Any printable characters can be included within the string. A special case within a string is the combination of two characters - "\" (backslash) followed by the letters "n", "t" or "f" - e.g. "\n". These combinations cause:
n- line break
t - tab with cursor movement to positions divisible by 8 (i.e. 0,8,16,24)
f - clear the screen and return the cursor to the top left

VAR
indicates a variable in which numerical values are stored. Since a 32-bit computer is used, all variables are also 32-bit. The values stored in them can be in the range -2,147,483,648 to 2,147,483,647. The user has a total of 126 automatically predefined variables. These are divided into two groups. Either they have their own name and are indicated by uppercase letters of the English alphabet A to Z (26 variables in total).
Or they form an indexed array of 100 items. The array can be accessed using the "@" character, immediately followed by the index number of the item given by one or two digits, or a letter indicating the variable where the index value is stored.

Examples of allowed variable names: A, .. Q, .. Z
Examples of array element names: @0, .. @9, .. @10, .. @28, .. @99, @A .. @Z.

Direct indexing of an array by digits does not exceed the array size, because the interpreter processes only one or two digits. However, if a variable is used for indexing, its contents may exceed the size of the array. For example, if A contains the value 4786, then a reference to an array element @A will cause an error message and the program to terminate.

EXPR
represents a mathematical expression with a computable value. It can be a number, the content stored in a variable, or an algebraic expression for calculating a specific value. The usable forms of expressions are described in the mathematics of the language below.

NUM
a number entered directly using consecutive decimal digits. The second option is to enter the number in binary form. A binary number is indicated by the letter "b" followed closely by the digits 0 or 1.

Mathematics of language

The language uses integer arithmetic, so it is not possible to enter numbers with fractional part. The range of a number can be from -2147483648 to 2147483647. Numbers outside this range will not be processed correctly.
Calculations process 32-bit numbers in the specified range. Addition, subtraction, multiplication and division operations can be used for calculations. The priority of operations (multiplication and division have priority over addition and subtraction) is respected. With the same priority level, expressions are processed from left to right.
Mathematics can be written formally:

```
        EXPR =  (  |+|-)TERM ((+|-) TERM)^
where   TERM = FACT ((*|/) FACT)^
and     FACT = VAR | NUM     .....  see above.
```

Items in parentheses are optional. Options are separated by the character "|". Parentheses marked "^" can be repeated until the end of the line.
Examples of mathematical expressions (EXPR) that are allowed:

```
-5
A/3 + @23
10 + A*15 - @G/42*R
-114783648 – Q * R* S* T + 526 / C
```

Parentheses cannot be used in mathematics in the program. If we need to calculate, for example, the formula A=(B+C)/(D-E), it is necessary to divide the calculation into several steps - for example:

```
LET F=B+C; LET G=D-E; LET A=F/G
```

Commands

The same formal description was used when entering the parameters of individual commands as the description given above for the language mathematics. Items in parentheses are optional and brackets marked "^" can be repeated until the end of the line.

Command: **PRINT**
Parameters: (STRING | EXPR) (,(STRING | EXPR))^
Command function: Prints the data specified in the parameters to the screen from the current cursor position. STRING is printed in text form without leading and closing quotation marks. EXPR is printed as a number expressed by the given mathematical expression.

Examples:

PRINT 5+3	prints the number 8 at the current screen position
PRINT "Result is " , B	prints according to the content of B e.g. Result is 115
PRINT A, "\n", B, "\n", C	will print the contents of variables A,B,C one after the other on the lines below them

---

Command: **IF**
Parameters: EXPR RelOp EXPR
Command function: This statement is closely related to the following THEN statement, which must immediately follow on the same line. They form a single, linked pair of statements, effectively representing a single double statement.

Command: **THEN**
Parameters: STATEMENT
Command function: Together with the IF statement, they create a decision condition that allows for conditional processing of another statement or branching of the program. The IF-THEN pair executes the STATEMENT statement in the THEN parameter only if the relational condition EXPR RelOp EXPR in the parameter of the IF statement is true.
RelOp is a comparison condition between two EXPR expressions. It can be of the form:

=	are both expressions the same?
<	is the first expression less than the second?
<>	are the expressions different?
<=	is the first expression less than or equal to the second?
>	is the first expression greater than the second?
><	are the expressions different?
>=	is the first expression greater than or equal to the second?
&	is the bitwise logical conjunction of the expressions non-zero?

If the condition is true, the STATEMENT command is executed, otherwise, the command is not executed.
Attention! If the condition is false, the execution of the line after the THEN keyword is not continued and the program moves to the next line of the program. This property can be used when executing multiple commands simultaneously. If the condition is met, the execution of the line continues and since there can be up to 3 commands on a line, it is possible to execute or not execute up to 3 commands after evaluating one condition.
Examples:

IF A>5 THEN PRINT A	the content of A is only printed if it is greater than 5

IF A*5=100 THEN GOTO 35	if the value in A is 20, the program jumps to line 35, otherwise it continues with the next line
IF C&b1000 THEN BEEP 1000,100	if bit no. 3 in variable C is log. 1, a tone will sound
IF A=0 THEN LET A=1; PRINT B	if the value of A is zero, A changes to 1 and B is printed

---

Command: **GOTO**
Parameters: EXPR
Command function: This is a jump function that transfers the current position of the processed lines to the line whose number is given by the expression EXPR. The command is only meaningful in the processing mode of a running program. If the user enters it as a command in the command entry mode, nothing is done. The command parameter is a general expression. However, the programmer must ensure that the value of this expression does not exceed the range 0 to 127. If an attempt is made to "jump" outside the allowed line number, an error is reported and program execution stops.

Examples:

GOTO 23	transfers program execution to line number 23
GOTO 10+I*5	according to the value in variable I, the program jumps to a line in multiples of 5

Command: **INPUT**
Parameters: VAR (,VAR)^
Command function: Entering numerical values into individual variables. The programmer can choose as a parameter which values the user should enter and these are gradually stored in the given variable. The user is gradually prompted to enter a number. When entering, the name of the variable being filled is displayed. For an array type variable, the user is prompted to enter only "@" (the index is not given). Numbers can only be entered in decimal form.

Examples:

INPUT L,H	the user is prompted to enter values into variables L and then H
INPUT @20, @38	the user will be asked twice to enter a value in @

Command: **LET**
Parameters: VAR = EXPR or @VAR = EXPR(,NUM)^
Command function: The assignment statement determines which EXPR value is inserted into the VAR variable.
The statement can also be used to simultaneously fill additional values into multiple array items. After entering a value into a specified array element, it is possible to enter numeric values separated by commas into subsequent elements. The number of values entered is limited only by the length of the line or by reaching the end of the array.

Examples:

LET I=52	the value 52 is written to I
LET Q= 23*W-L/3	the expression 23*W-L/3 is first calculated and the result is written to

	Q variable
LET @23=4,12,8,63,47	array elements 23 to 27 are filled with the values entered sequentially

---

Command: **GOSUB**
Parameters: EXPR
Command function: Calling a subroutine, which can be on reserved program lines. This command is very similar to the GOTO command and does practically the same thing. In addition, however, the return value for the RETURN command is stored. The return value is the line number 1 greater than the line on which the GOSUB command is currently being processed. After executing the subroutine, the RETURN command returns the program to the line given by the return value. Within a subroutine, it is possible to call another subroutine and thus nest subroutines within each other. However, the maximum level is limited to 8 nested subroutines. An attempt to call the ninth subroutine from the eighth nested subroutine will cause an error message and the program to terminate.

Examples:

GOSUB 44	the subroutine stored on line 44 will be called, after it is terminated with the RETURN command, it will continue on the next line

---

Command: **RETURN**
Parameters: ---
Command function: Terminate a subroutine. The statement takes the return value stored by the GOSUB command and places it in the pointer to the next line to be executed. Calling the RETURN statement without first calling the GOSUB statement (i.e., missing a return value) will cause an error to be reported and the program to terminate.

Examples:

RETURN	ends the subroutine and returns the program to the line following the GOSUB command

---

Command: **CLEAR**
Parameters: ---
Command function: Delete the current program. After executing this command, all lines of the program will be empty. If any program was written in the current file (see FILE command), it will be completely deleted.

Examples:

CLEAR	deletes the current program

Command: **LIST**
Parameters: (NUM)
Command function: Prints the current program in memory. The command can be given without a

parameter. In this case, it prints the entire current program stored in the computer's memory. If a number is specified as a parameter, it prints only a specific line. The number can be any, but only the lowest 7 bits are valid (so-called modulo 128 arithmetic).

Examples:

LIST	prints the entire program on the screen
LIST 15	prints line 15 of the program
LIST 130	prints line 2 of the program

Command: **RUN**
Parameters: ---
Command function: Starts program execution. After starting, the program lines are executed sequentially from line 0. If there is no jump command (GOTO or GOSUB) on the given line, the line with a number 1 higher is executed after the current line is executed. Lines on which no command is stored are searched, but no action is performed on them. They represent only a minor delay in the program (the line must be read). For time-critical parts of the program, it is advisable not to leave lines unnecessarily empty. If the line pointer exceeds the value from 127 to 128, the program is automatically terminated.
A special case is writing the RUN command in the file FILE 0 (on the disk "h") at the first position of line 0. If the computer finds the RUN command at this position, it automatically starts this program after it is turned on. The computer does not need to have a keyboard connected and can be used, for example, to control a device after debugging the program.

Examples:

RUN	runs the written program

Command: **END**
Parameters: ---
Command function: Terminates program execution. If the END command is processed during program execution, the program is terminated and the computer returns to the waiting for user command mode. The END command is the only command whose pressing of the appropriate keys (Ctrl + N) is tested during the transition between lines during program execution. When this command is entered on the keyboard, the running program is terminated.

Examples:

END	ends the running program

Command: **TIME**
Parameters: VAR
Command function: It stores the value from the permanently running internal counter in the variable VAR. The counter is incremented by 1 every time a graphic line is drawn on the VGA monitor since the computer started. To synchronize with the oscillator frequency, the counter runs at a frequency of 31914 Hz. This means that its content is incremented by 31914 every second. Since the counter

is 32-bit, it must overflow. After reaching the value 2,147,483,647, which will happen after about 18 hours of operation, the value jumps to the smallest negative number -2,147,483,648 and continues to increase by 1. Further overflows will then always occur after more than 37 hours.
The function can be used not only to monitor the passage of a certain time interval during program execution, but also, for example, as a random number generator. When you press a key and save the current time, you just need to calculate the remainder after dividing the time by the number that determines the generator range.

Examples:

TIME T	stores the time in the variable T
TIME R; LET Q=R/6; LET R=R-Q*6	in the variable R we get a random number from 0 to 5 (e.g. for a dice roll)

Command: **CURSOR**
Parameters: EXPR
Command function: Sets the cursor (the place where writing will be done on the screen) to the specified position. The upper left corner of the screen is position 0 and the lower right corner is position 799. Positions increase on lines from left to right, just as when writing. If the position is outside the allowed range, the command does nothing and the cursor position remains at its original value.
If it is necessary to set the cursor position using a line number, then a calculation can be used where the line number is multiplied by 32 and the character positions within the line are added.

Examples:

CURSOR A	moves the cursor position to the values according to variable A
CURSOR 32*5+15	moves the cursor to the sixth line from the top and the 16th position on that line
CURSOR 1024	does nothing (value is out of range)

Command: **PUTCH**
Parameters: EXPR or x EXPR (where x see next) or p EXPR, EXPR
Command function: At the position specified by the cursor (the place where the screen will be written), a character is printed whose value is determined by the parameter expression. The lowest 8 bits are used for the printout. The character is printed in the range 0 to 255 (for higher numbers it will be the remainder after dividing by 256). The drawn character corresponds to the ASCII table. For example, the letter "A" has a value of 65.
Within the ASCII table range, only printable characters from 32 to 127 will be displayed. Above this range, semigraphic characters (mainly lines suitable for various frames) will be displayed in the range 128 to 159. After the semigraphic characters there is an area whose value is expanded as a BASIC keyword. For example, the value 160 will cause the word "PRINT" to be printed.
The remaining characters are non-printable and a dot will appear instead. The only exceptions are 3 values that correspond to the characters "\n", "\t" and "\f", as mentioned for the STRING parameter above:

9	"\t" - move the cursor on the line to the next higher tab position 0,8,16 or 24
10	"\n" - newline

12 "\f" - clears the screen and places the cursor at position 0

The cursor moves in the same way as when printing text using the PRINT command. At the end of a line, a new line is entered and, if necessary, the entire screen is moved.

If one of the following characters is given before EXPR, the cursor moves in a special way:

. the cursor will remain at the same position
> the cursor moves one place to the right
< the cursor moves one place to the left
v the cursor moves down one place
^ the cursor moves up one place

In these cases, the line or column is not moved to a new line or column. The cursor moves (and overflows) within the same line or column. In this way, it is possible to print only printable characters in the range 32 to 159.

The last option is to insert a character directly at the specified position on the display without affecting the cursor. After the "p" character, the value of the EXPR character is given, which is inserted at the position specified by the second parameter. The position on the display is in the range 0 to 799, the same as with the CURSOR command. In this way, it is again possible to print only printable characters in the range 32 to 159.

Examples:

PUTCH 49	prints the number 1 at the cursor position
PUTCH p68, 32*12+10	prints the letter D on line 12, position 10
PUTCH 12	clears the screen and sets the cursor to the upper left corner
PUTCH .65	prints the letter A and leaves the cursor at the same position
PUTCH v67	prints the letter C and moves the cursor to the lower line with the same position (in the case of the lowest line, it goes to the highest line)

Command: **GETCH**
Parameters: VAR or pEXPR, VAR
Command function: It stores the value of the character located at the cursor position on the screen in the variable VAR. Using the CURSOR command, it is possible to set the cursor to any position and using the GETCH command, it is possible to search for what was written at that position.
The second option is to specify the position on the display directly after the character "p" in the range 0 to 799, from which the character is taken and stored in the variable VAR.

Examples:

CURSOR 392; GETCH C	the character from position 12,8 on the screen is inserted into variable C
GETCH p32*15+4, L	the character from position 4,15 on the screen is inserted into the variable L

Command: **INKEY**
Parameters: VAR
Command function: The command tests whether the user has pressed a key. If no key has been pressed, the value 0 is stored in the variable VAR. If a key has been pressed before the command is called, the corresponding character value according to the ASCII table is stored in the variable. For example, the value 66 means that the user has pressed "B".

Examples:

INKEY C	the character of the key already pressed is inserted into the variable C, or 0 if nothing was pressed

---

Command: **FILE**
Parameters: (EXPR)
Command function: Depending on the size of the connected memory, the user has at least 1 and at most 16 files available for writing a 128-line program. Switching between these programs is done using the FILE command, where the EXPR expression determines which file (program) will be active. This program is then worked with, written to or read from. EXPR can take on any value, but only the lowest 4 bits are valid (the remainder after dividing by 16). So, for example, the value 17 is equivalent to the value 1.
The size of the memory used determines how many files the user has available. If the smallest memory 24C32 is used, the FILE command has no meaning, because only the file with number 0 is available and it is still active. The larger memory 24LC64 already has two files – 0 and 1. The EXPR expression then allows switching between files, depending on whether the expression is odd or even. With the 24LC128 memory, you can use EXPR to sequentially select files 0,1,2,3,0,1,2,3,0,1... Only the 24LC512 memory uses the full range of this command and selects files numbered 0 to 15.
The FILE command will probably be used most often when changing programs by the user. It can also be used within a running program. This provides a powerful tool that allows you to link different files and jump between them on the fly.
For example, if a program is running in file number 0 and executes the command FILE 3 on line 25, the program continues with line 26 in file 3. This way, the program size limitation of 128 lines can be bypassed. When using the 24LC512 memory and using the option to write up to 3 commands per line, a program of up to 6000 commands can theoretically be placed in memory. In combination with the DISK command (see below), this number increases to 12000.
If the parameter is not specified (or the expression is entered incorrectly), this command prints the current file number on a new line.

Examples:

FILE 10	switches to file number 10 (for 24LC512 only), to file 0 (for 24C32, 24LC64) and to file 2 (for 24LC128 and 24LC256)
FILE	prints the current file number

Command: **DISK**
Parameters: (h|f)
Command function: The command switches between the memories on the computer board. One of the memories is SMD, firmly soldered to the board and cannot be changed. This memory represents the computer's "hard disk" and is marked “h”. The other memory in DIL can be inserted and removed from the socket and is marked “f”. The integrated circuit can be marked as a "floppy disk" - a removable medium. After the computer starts, DISK h is active and the file FILE 0 in it.
The parameter determines which disk will be used next – it will be active.
If DISK f is not inserted at startup, the computer will report this in the initial message after switching on. Then, of course, it is not possible to switch to this disk using the DISK f command. The command does nothing and does not report an error. DISK h will remain active.

If the parameter is not specified or is chosen incorrectly, the command will print the active disk designation on a new line.

Examples:
DISK f switches to DISK f if inserted; otherwise does nothing
DISK lists active DISK

Command: **DELAY**
Parameters: EXPR
Command function: Pauses the program for the time specified by the EXPR parameter. The value of the expression specifies the time in milliseconds. Allowed values are 1 to 1,000,000. Outside this range, the command does nothing.

Examples:
DELAY 5*100 the program waits 500 milliseconds and then continues

---

Command: **BEEP**
Parameters: EXPR, EXPR
Command function: A small speaker on the board will emit a tone with a frequency specified by the first parameter for a duration specified by the second parameter. The first parameter specifies the frequency in Hz and the allowed values are in the range of 100 to 4000 Hz. Given the parameters of these small sound buzzers, different frequencies will be different loudness and a larger range of values does not make sense.
The second parameter defines the duration for which the tone will sound in milliseconds. Allowed values are between 20 and 100,000 milliseconds. If the parameter range is not respected, the command will not be executed – nothing will be done.

Examples:
BEEP 800, 1500 an 800 Hz tone will sound for 1.5 seconds

Command: **REM**
Parameters: Any text
Command function: An empty statement that does nothing. It serves only as a comment for the programmer, who can write a short note in the program. The text up to the end of the line is considered a comment.

Examples:
REM Next calculate area of circle any text can be up to the end of the line.

Command: **COPY**
Parameters: (h|f)NUM or sNUM,NUM,NUM
Command function: Allows you to copy program source text between files or copy lines within the

active file.
The first option causes the contents of the active file will overwrite the file specified in the parameter. The letters h or f specify the disk to which the active file will be written, and the number NUM specifies the target file on that disk. All lines 0 to 127 are copied completely. An exact copy of the file on the given disk is created.
The second parameter specification, indicated by the letter s, copies lines within the active file. The first number (from 0 to 127) specifies the source line where copying will begin. The second parameter (from 0 to 127) selects the location where saving will begin. The third parameter (1 to 127) then specifies how many lines will be copied. If the end of the file is reached during copying, the copying will stop.

Examples:

COPY f6	copies the active file on disk f to file no. 6
COPY s25,50,20	copies 20 lines in the active file from line 25 to lines 50 and above

---

Command: **AINP**
Parameters: VAR
Command function: The computer has one analog input as standard, allowing you to measure voltage in the range of 0 to 5V. A voltage of 0V is measured as a value of 0, and a maximum voltage of 5V corresponds to a value of 1023. It is important to remember that this is not a precise measuring instrument. The measurement range and accuracy are affected by several factors. However, for ordinary amateur use, this should be sufficient. The value measured by the AINP command is stored in the VAR variable.
Voltages outside the specified range cannot be measured and such voltages can very likely damage the computer!

Examples:

AINP @32	measure the analog voltage at input A and store it in the array element with index 32

Command: **DINP**
Parameters: VAR, (1|2) - for version with crystal, without crystal: VAR, (1|2|3|4)
Command function: When using a crystal as a source of precise frequency, the computer has 2 digital inputs prepared on the board. If the crystal is not installed and the computer is clocked by an internal oscillator, the number of inputs is expanded to 4. The command reads the state of the input with the specified number into the variable VAR. The value stored in the variable can be either 0 or 1 depending on the logic level at the input. An unconnected input behaves as a logic 1. Using the command switches to the input mode with a weak pull-up to log 1. The pin remains as an input until the DOUT command is used.

Examples:

DINP R,2	the logical state of digital pin 2 is read into variable R

Command: **DOUT**
Parameters: EXPR, (1|2) - for version with crystal, without crystal: EXPR, (1|2|3|4)
Command function: The opposite command to the previous DINP command. After its execution, the given digital pin becomes an output with a specified logic level. The command writes the value EXPR to the specified output. If the value is 0, then the output is also logical 0. Any other value then writes a logical 1 to the output. Until the DINP command is used, the pin remains as an output.

Examples:
DOUT 23,2 the output is set to logical 1 (value is not 0) on digital pin 2

Command: **I2CW**
Parameters: EXPR
Command function: The computer has relatively few digital inputs and outputs (2 or 4 depending on the board configuration). If this number is not enough for your project, the inputs and outputs can be expanded using a PCF8575 expander board. Then the number is expanded by another 16 digital inputs or outputs. The expander can be connected using a connector with an I2C bus on the board.
Ready-made boards can often be purchased as accessories for Arduino modules, etc. Be careful with the connection of the connectors on the expander board! The PCF8575 board can usually be plugged directly into the connector, but check your expander board!
The command sends the lower 16 bits of the EXPR expression to the expander board. It therefore makes sense to send only a value up to max. 65535.

Examples:
I2CW 61455 The middle 8 bits in the PCF8575 expander will be 0 padded with 1 from both the top and bottom.

Command: **I2CR**
Parameters: VAR
Command function: An additional command to the previous I2CW command – see above. It allows reading the input status from the PCF8575 expander board into the VAR variable.
Again, the standard assumes a 16-bit expander PCF8575. It reads the value up to a maximum of 65535.

Examples:
I2CR H reads the status from the PCF8575 expander into variable H

Example program

As an example of a short program, the procedure for displaying all printable characters on the screen was chosen:

```
0  REM  Printable characters
1  PUTCH  12;  LET  A=32
2  PUTCH  A;  LET  A=A+1
3  IF  A<160  THEN  GOTO  2
```

Line 0 is just a comment with the program name and does nothing.
Line 1 first clears the screen by sending the character 12 and then initializes the variable A to 32.
Line 2 prints the character from A to the display and increases the value of variable A by one.
Line 3 tests whether the final value has already been reached and if not, then it returns to line 2.

This program allows you to display printable characters on the screen. When you run it, 4 lines of characters appear. The lines start with a character with a value corresponding to 32, 64, 96, and 128.

What to do when...

When using a computer, situations may arise that cause the device to freeze. In these cases, the following procedure may help:

1) If a running program contains an infinite loop, it is possible to interrupt its execution by pressing the RESET button. This performs a so-called cold start on the computer, when all variables are set to zero.
   The second option is to enter the END command (Ctlr+N) during program execution. The program interrupted in this way leaves the value of the variables as they were at the moment of interruption. Entering the END command usually leaves this command or only the letter "n" on the command line. It is necessary to additionally delete the rest of the command.
2) If the connected keyboard is manipulated (disconnected and connected) after the computer is turned on, its communication may be disrupted. The keyboard then does not respond or enters incorrect characters. The problem can be solved by pressing the RESET button.
3) The computer has programs stored in a serial EEPROM memory connected to the I2C bus. In case of any interfering signals on the bus, the memory can be blocked and the computer stops responding completely. Often, pressing the RESET button does not help either. In this case, it is necessary to disconnect the computer from the display and turn off the power. After a few seconds in the off state, the computer can be turned on again and connected to the display.
4) **Always insert or remove memory from the "DISK f" socket when the computer is turned off (without power)!** Hazardous bus conditions when manipulating the memory can damage its contents or the contents of the memory representing the "DISK h". A similar approach is required when connecting or disconnecting an expander with the PCF8575 circuit.

Commands and parameters:

Command	Ctrl +	Parameters	Note
PRINT	**P**	(STRING | EXPR) (,(STRING | EXPR))^	
IF	**U**	EXPR RelOp EXPR	RelOp = 7 different operators
THEN	**T**	STATEMENT	
GOTO	**G**	EXPR	0 to 127
INPUT	**I**	VAR (,VAR)^	
LET	**L**	VAR = EXPR or @VAR = EXPR(,NUM)^	
GOSUB	**H**	EXPR	0 to 126
RETURN	**Y**		
CLEAR	**X**		
LIST	**K**	(NUM)	Use modulo 128
RUN	**R**		
END	**N**		
TIME	**V**	VAR	31914 Hz ticks
CURSOR	**W**	EXPR	0 to 799
PUTCH	**C**	EXPR or x EXPR or p EXPR, EXPR	x choose / p Character, Position
GETCH	**,**	VAR or p EXPR, VAR	p Position, Variable
INKEY	**S**	VAR	
FILE	**F**	(EXPR)	Modulo 16
DISK	**D**	(h|f)	
DELAY	**M**	EXPR	1 to 1000000
BEEP	**Z**	EXPR, EXPR	Frequency, Time
REM	**E**	Any text	
COPY	**.**	(h|f)NUM or sNUM,NUM,NUM	s Source, Destination, Number
AINP	**A**	VAR	
DINP	**B**	VAR, (1|2) or VAR, (1|2|3|4)	Crystal / without crystal
DOUT	**O**	EXPR, (1|2) or EXPR, (1|2|3|4)	Crystal / without crystal
I2CW	**Q**	EXPR	16 bits
I2CR	**J**	VAR	16 bits

^ - repeat option

| - choose from alternative options