2. The Structure of IC Libraries

To avoid creating a new file for every IC it is possible to combine several ICs in one library. The directories can then be kept as small and clear as possible.
An IC library is a 64 kbyte-sized file which can be read and modified by every standard text editor. The data for the individual ICs is subdivided into five blocks:

1. Name of the IC
2. Information text of the IC
3. Pin number and pin arrangement
4. Family (Standard, LS, L, HC ...)
5. Test algorithm

The first four points are pure information blocks which appear on the screen during testing or when the user demands information.
The fifth point, the test algorithm, is the actual IC-PASCAL test program and is executed by the integrated IC-PASCAL interpreter when required; consequently it tests out the electronic components.
Each of the five blocks is assigned a key name beginning with a "#":

1. #NAME
2. #TEXT
3. #PIN
4. #FAMILY
5. #PROGRAM

An example should clarify the subdivision of the four information blocks:

The IC, type SN7400 (4 x NAND), is to be programmed.

```
#NAME 7400,SN7400
```

IC terms, separated by commas, come after "#NAME". By entering the appropriate name the program then searches for the IC.

```
#TEXT
The IC SN 7400 contains four
separate NAND gates each
with two inputs.
```

After the "TEXT" statement there normally follows a short function description of the respective IC. This is inserted top left in the IC test software.

```
#PIN 14
 1 : Input   1 Gate   1
 2 : Input   2 Gate   1
 3 : Output    Gate   1
 4 : Input   1 Gate   2
 5 : Input   2 Gate   2
 6 : Output    Gate   2
 7 : GND
 8 : Output    Gate   3
 9 : Input   1 Gate   3
10 : Input   2 Gate   3
11 : Output    Gate   4
12 : Input   1 Gate   4
13 : Input   2 Gate   4
```

The number of pins of the IC follows "#PIN"; in this example 14. If necessary pin descriptions can be given appearing at the bottom right of the IC tester software. The number of the pin, then a colon and finally the text must be defined.

```
#FAMILY Std,ALS,F,H,L,LS,S
```

'#FAMILY' is the command to define the IC family. This

can be displayed as required
using the function "FAMILY".

```
#PROGRAM

BEGIN
...
...
END.
```

After '#PROGRAM' the above mentioned test program must follow which we will now look at in greater detail.

These five points can be placed after one another in an IC library for different ICs as long as their combined length does not exceed 64 kbyte.

3. Writing Programs in IC PASCAL

IC-PASCAL places over 70 key words and symbols at your disposal enabling test algorithms for integrated circuits to be written.

Every program in IC-PASCAL starts with the key word "BEGIN" and ends with "END."; a sequence of IC-PASCAL statements, separated from each other by a semi-colon, is placed between these two markers. Those who know PASCAL should note that IC-PASCAL, version 1.3, does not allow definitions of procedures or functions; neither can the user define his own variable names. IC-PASCAL itself provides 26 base variables plus an array containing 256 variables and a maximum of 20 pins can be addressed. IC-PASCAL operates with "Unsigned Long Integer"-numbers (32 bit) giving the possibility of computing integral values from 0 to 4294967295.

Let's look at a few short examples which will help you to get to grips with IC-PASCAL. Using your text editor create the file "TEST.IC". In future we will write our test programs in this file.

3.1. The Fundamentals of IC PASCAL

Please enter the following text:

```
#NAME TEST1

#PROGRAM

BEGIN
   WRITE ('IC-Tester V1.3');
   WAIT;
END.
```

Store this file on a disk or hard disk. Now exit the text editor program and load this file into the IC tester software using the function "Load". You can now start the program at any time with the function "Test" and by entering the name "Test1". The IC-PASCAL-Interpreter will then open a small window in which 'IC-Tester V1.3' appears. Thus the statement 'WRITE' opens the window and writes the following text. The 'WAIT'statement keeps the text on the screen; the interpreter closes all windows after a program has been executed and so the text would only be visible for a fraction of a second. Naturally the information text and pin arrangement window remains blank during the program execution as there is no data available for this (#TEXT or #PIN !). For the same reason nothing will be

displayed via the menu function 'Family' (#FAMILY !).

Now extend the file 'TEST.IC' with the following program:

```
#NAME TEST2

#PROGRAM

BEGIN
   WRITE ('IC-Tester V1.3');
   WRITE ('------------------');
   WAIT;
END.
```

After starting this program (please observe that you must now enter 'TEST2') you will notice that the intepreter has not done what one might expect: both texts are read out next to each and not under one another. The reason for this is that the 'WRITE' command leaves the cursor after the first text and does not position it at the beginning of the next line. This can be remedied with the 'WRITELN' command (Write Line), which transports the cursor to the beginning of the next line.
In addition several expressions can be written out with a single 'WRITE'- or 'WRITELN' command. The individual sequences are separated by a comma, e.g. WRITELN (1+2,3+4,5+6).
Perhaps you have already noticed how important the semi-colon is after statements; without this statement separater the intepreter cannot distinguish between two statements and indicates 'Syntax Error'.

3.2. Simple Arithmetic Expressions

As well as text expressions, arithmetic expressions can also be displayed on the screen using the 'WRITE'- or 'WRITELN' statement. Please enter the following example:

```
#NAME TEST3

#PROGRAM

BEGIN
   WRITELN (100);
   WRITELN (100+100);
   WRITELN (100-100);
   WRITELN (100*100);
   WRITELN (100 DIV 100);
   WAIT;
END.
```

You will see that the computer tries to evaluate texts which are not set in inverted commas. If certain computation rules are not observed this can easily lead to program errors (e.g. division by zero).
Please note that in IC PASCAL the word 'DIV'is used for division and not the oblique line '/'. The interpreter only carries out integer division because, as mentioned above, it can only process integer numbers; i.e. 5 DIV 2 equals 2 and not 2.5; 5/2 however, must result in 2.5.
You can obtain the division remainder by employing the operator 'MOD'; 5 MOD 2 equals 1 (5-(5 DIV 2)*2=1 !).

The normal algebriac conventions apply to IC-PASCAL, so multiplications and divisions take priority over additions and subtractions. In the case of 3+4*5 , four is multiplied by 5 and then 3 is added to this product: 3+4*5 = 3+(4*5) = 3+20 = 23.

Note:
Numerical values can be processed binary, hexadecimally or decimally. A % symbol must be placed before binary number values and a $ symbol before hexadecimal number values. Decimal number values are not specially marked. A numerical example illustrates this:

```
%01010101 = $55 = 85
```

3.3. Variables in IC-PASCAL

All 26 letters of the alphabet are predefined as variable names in IC-PASCAL; redefinitions are not possible.

A variable can be compared with the memory of a pocket calculator: intermediate values can be filed until further processing or for an indefintie period.
These variables can of course only accept numbers out of the numerical range in which IC-PASCAL is capable of working; i.e. integer values between 0 and 4294967295 are ascertainable.

In PASCAL the assignment of simple numbers and arithmetic expressions is generally undertaken with ':='. In contrast to several other programming languages, which work with '= ', this indicates that this sequence has nothing to do with a mathematical equation but solely concerns the assignment of an expression to a variable.

Look at the following example:

```
#NAME TEST4

#PROGRAM

BEGIN
   A:=3;
   B:=4;
   C:=5;
   WRITELN (A+B*C);
   WAIT;
END.
```

In the program TEST4 the interpreter assigns the values 3, 4 and 5 to the variables A, B and C. Then B and C are multiplied (see 3.2) and A is added to this product giving a result of 23.
If you have slightly more difficulty working with variables and arithmetical terms then try to change the last two program examples a little. You will soon see - practice makes perfect!

If the 26 variables prove to be insufficient the user can turn to an array containing 256 variables having the name 'MEMORY' or simply 'MEM'.
The variables (the value range corresponds to the variables A-Z) are numbered in series from 0 to 255. This number is written in square brackets behind the word 'MEMORY' or 'MEM'; for example MEM[100] or MEM[125].

The following example illustrates this:

```
#NAME TEST5
```

```
#PROGRAM

BEGIN
   MEM[0]:=3;
   MEM[1]:=4;
   MEM[2]:=5;
   WRITELN (MEM[0]+MEM[1]*MEM[2]);
   WAIT;
END.
```

3.4. Repetitive Statements in IC-PASCAL

These are statements which permit parts of programs to be executed repetitively. Owing to their nature they are called loops. IC-PASCAL offers three typical statement structures allowing us to construct such sequences. Out of these three the 'FOR-TO-DO'loop is probably the easiest to understand.

3.4.1 The FOR-TO-DO Loop

This type of loop is not only the simplest but also the most widely used of loop structures. A program loop of this sort is found in almost all languages from BASIC to C; even if the statements are somewhat different the basic principle is the same.

In each case a variable is necessary for the execution of the FOR-TO-DO loop, the so-called 'control variable'. This variable then 'counts`, for example from 10 to 25 and executes a command each time.
The following example clarifies this process:

```
#NAME TEST6

#PROGRAM

BEGIN
   FOR I:=10 TO 25 DO
      WRITELN (I);
   WAIT;
END.
```

The very first thing that the program TEST6 does is assign the value 10 (the initial value) to the variable I. Then the statement WRITELN (I) is executed. Following this I is incremented by one and compared to 25. If I is less than or equal to 25 the intepreter executes the WRITELN (I) command again and so on until I exceeds 25; the final value. The numbers 10 to 25 then appear as the result on the screen.

It is also possible to allow a variable count down using the statement 'DOWNTO':

```
#NAME TEST7

#PROGRAM

BEGIN
   FOR I:=25 DOWNTO 10 DO
      WRITELN (I);
   WAIT;
END.
```

In TEST7 the variable I is checked to see if it is greater than or equal to the final value, if it is the variable is decremented

by one. The numbers 25 to 10 then appear on the screen as the result.

A special feature in IC-PASCAL allows the step length for increasing or decreasing to be determined. This value is set with the 'BY'command as in 'MODULA2', the successor of PASCAL.
The following example illustrates this:

```
#NAME TEST8

#PROGRAM

BEGIN
   FOR I:=0 TO 25 BY 5 DO
      WRITELN (I);
   WAIT;
END.
```

The program example TEST8 reads out all numbers from 0 to 25 with a step extent of 5; thus 0, 5, 10, 15, 20 und 25.

Please note that not only the variable I can be used as control variable but every available variable in IC-PASCAL!

As mentioned above the interpreter only carries out one command with the FOR-TO-DO statement; to execute a whole sequence of statements in this form the statements must be bracketed with the markers 'BEGIN' and 'END'.

By surrounding one or more statements with 'BEGIN'and 'END' the interpreter judges this to be one statement (compound statement) and executes it accordingly. This is illustrated in the following example:

```
#NAME TEST9

#PROGRAM

BEGIN
   FOR I:=1 TO 10 DO
      BEGIN
         WRITELN (I);
         WRITELN (I);
      END;
   WAIT;
END.
```

The program example TEST9 writes each number from 1 to 10 twice, one below the other. If the markers 'BEGIN'and 'END' were omitted, then the numbers would only be written once and the last one twice.

3.4.2. The WHILE-DO Loop

In the WHILE-DO loop the statement following 'DO' is executed as long as the specified condition between 'WHILE' and 'DO' is true.

Conditions in IC PASCAL demand a comparison operator.
The following six comparison operators are available:

=	is equal to
<>	is not equal to
<	is less than

```
           <=    is less than or equal to
           >     is greater than
           >=    is greater than or equal to
```

A comparison operator is always between two arithmetic expressions. The result of this comparison is either TRUE or FALSE, the condition is accordingly fulfilled or unfulfilled.
The following example will help you to see how to work with operators of this kind:

```
Condition fulfilled (true)  Condition not fulfilled (false)

1 = 1                        1 <> 1
1 < 2                        1 > 2
1+2 = 3                      1+2 <> 3
5-2 = 9-6                    5-2 <> 9-6
1 <> 2                       1 = 2
9 >= 2                       9 < 2
```

As well as figures (constants) variables can also be inserted as values and compared with one another.

Instead of using the FOR-TO-DO statement used in program example TEST6 you can employ the WHILE-DO loop type:

```
#NAME TEST10

#PROGRAM

BEGIN
   I:=10;
   WHILE I<=25 DO
      I:=I+1;
   WAIT;
END.
```

First of all the variable I is assigned a value of 10. Then I is compared to 25; if I is less than or equal to 25 the statement following 'DO'is executed thus I is incremented by 1 (I is given the value I+1). The interpreter then compares I with 25 again and so on. The loop is terminated when I has reached a value of 26.

If you wish to have the value of the variable I displayed on the screen during the counting process then the statement WRITELN (I) must be added. However, as only one expression following DO can be executed in a WHILE-DO loop the statement I:=I+1 and the command WRITELN (I) must be bracketed between the marker words BEGIN and END:

```
#NAME TEST11

#PROGRAM

BEGIN
   I:=10;
   WHILE I<=25 DO
      BEGIN
      WRITELN (I);
      I:=I+1;
      END;
   WAIT;
END.
```

3.4.3. The REPEAT-UNTIL Loop

In the REPEAT-UNTIL loop the statements between REPEAT and UNTIL are obeyed until the condition following UNTIL is true.

In the following example the FOR-TO-DO loop used in TEST6 is substituted with a REPEAT-UNTIL loop:

```
#NAME TEST12

#PROGRAM

BEGIN
   I:=10;
   REPEAT
      WRITELN (I);
      I:=I+1;
   UNTIL I>25;
   WAIT;
END.
```

The program TEST12 allocates a value of 10 to the variable I. Following this the WRITELN command prints I on the screen and it is incremented by one with the sequence I:=I+1, then the condition following UNTIL is tested. If this condition is false the interpreter returns to the REPEAT instruction. The REPEAT loop terminates when I has a value of 26.
Several statements may be placed inside the loop without BEGIN and END since, as mentioned above, all commands between REPEAT and UNTIL are executed every time the loop is repeated.

3.5. Selective Program Execution

Using the 'IF-THEN' statement particular program segments can only be executed if a defined condition applies.
This command in PASCAL is comparable to the identical sequences in BASIC or C.

When the condition between IF and THEN is true the statement following THEN is executed, otherwise the next statement, if is there is one, is carried out.

Look at the following example:

```
#NAME TEST13

#PROGRAM

BEGIN
   FOR I:=1 TO 100 DO
      IF I=50 THEN WRITELN (I);
   WAIT;
END.
```

In TEST13 the FOR-TO-DO loop is iterated 100 times. Every time it is repeated the IF-THEN command checks whether the variable I has a value of 50; if this is the case then the WRITELN statement prints I on the the display unit and the loop continues.

By adding 'ELSE'to the IF-THEN statement it is possible to select one of two statements by evaluating a specified condition. In the following example notice that there is no semi-colon after the statement following THEN :

```
#NAME TEST14
```

```
#PROGRAM

BEGIN
   FOR I:=1 TO 100 DO
      IF I=50 THEN WRITELN (I)
              ELSE WRITE ('*');
   WAIT;
END.
```

In TEST14, as in TEST13, the FOR-TO-DO loop is repeated 100 times. The IF-THEN statement checks every time if the variable I has a value of 50; if this is the case then, as in the previous example, I is written up on the screen and the next loop iteration is undertaken. Otherwise the command following 'ELSE'is executed and an '*' appears on the screen.

3.6 Direct VDU Control

In the previous program examples we saw how the IC-PASCAL interpreter opens a small output window when the WRITE or WRITELN command is used for the first time. From that point on the output text appears in this window.
IC-PASCAL takes this function a step further by incorporating VDU control instructions which permit the output window to be enlarged or reduced, a cursor to be switched on or off or the output to be shown in particular positions only:

CLRSCR	(Clear Screen) clears the screen
WINDOW (1,1,20,20)	opens the output window at a defined position (here column 1, line 1 to column 20, line 20)
GOTOXY (5,6)	positions the cursor at a particular column and line (here column 5, line 6). The output is given at this position.
HOME	positions the cursor at the top left in the output window.
CURSORON	switches on cursor
CURSOROFF	switches off cursor

4. Programming Test Algorithms

The actually task of IC-PASCAL is to enable the programming of test algorithms for integrated circuits.
With this aim in mind statements are available to define voltage-carrying pins, input and output pins etc.

In IC-PASCAL the dummy array 'Pin' manipulates the individual pins of the test socket. PIN[1] represents pin 1, PIN[2] represents pin 2 and so on. The important thing is that the number of the pins being manipulated is defined with '#PIN' (see above) !

Different states can be assigned to individual pins with a ':'.

+5V	the pin carries a supply voltage of +5V

GND	the pin carries ground
INPUT	the pin is an input of the IC and can be set to LOW or HIGH
OUTPUT	the pin is an output of the IC and can supply either LOW or HIGH
RC	the pin is connected with two other pins in an RC combination (to test (mono-flops etc.)
LOAD LOW	the pin is negatively loaded (to test OPEN COLLECTOR ICs etc)
LOAD HIGH	the pin is positively loaded (to test OPEN COLLECTOR ICs etc.)
+5V OFF	switches off the supply voltage
GND OFF	switches off the ground arrangement
RC OFF	switches off the RC combination

Firstly, the current-carrying pins and the inputs and outputs are defined. In the example below this is carried out on one gate of the SN 7400 (4 x NAND):

```
#NAME 7400,SN7400

#PIN 14

#PROGRAM

BEGIN
PIN[7] : GND;
PIN[14] : +5V;
PIN[1] : INPUT;
PIN[2] : INPUT;
PIN[3] : OUTPUT;
...
...
END.
```

In this case pin 7 is defined as earth-carrying, pin 14 as having +5V, pins 1 and 2 as inputs and pin 3 as output. We can now assign LOW or HIGH to the pins 1 and 2 and check whether pin 3 gives the NAND operation of these two inputs. IN IC-PASCAL the allocation of LOW or HIGH is always made via ':=', for example PIN[1]:=LOW;PIN[2]:=HIGH etc. Four different combinations must be tested when there are two inputs:

PIN[1]	PIN[2]
LOW	LOW
LOW	HIGH
HIGH	LOW
HIGH	HIGH

It would take a lot of time to allocate each of these combinations individually to PIN[1] and PIN[2] and then check whether PIN[3] also corresponds to the NAND operation of these pins. For this reason IC-PASCAL offers the possibility of using a program loop to carry out this.
LOW and HIGH states can also be written as 0 and 1. If we now allow a variable, for example I, to run from 0 to 3 the two lower bits of this variable will yield the combination shown above:

I	Bit 1	Bit 0

```
          -----------------
          0     0       0
          1     0       1
          2     1       0
          3     1       1
```

The assignment of individual bits to pins results using the '&'command (sequential assignment), therefore

```
          PIN[1] & PIN[2]:=I;
```

This instruction 'spilts'the variable into its individual bits; the last variable receives the least significant bit of I and so on.

In the same way the state of several pins can be 'packed together' with the '&' command:

```
          I:=PIN[1] & PIN[2];
```

After this sequence has been executed the first listed variable in I is the most significant bit, the last variable the least significant bit.

Thus the program to test SN 7400 must look like the following: (testing one gate only !)

```
#NAME 7400,SN7400

#PIN 14

#PROGRAM

BEGIN
PIN[1] : INPUT;
PIN[2] : INPUT;
PIN[3] : OUTPUT;
PIN[7] : GND;
PIN[14] : +5V;

FOR I:=0 TO 3 DO
   BEGIN
   PIN[1] & PIN[2]:=I;
   IF PIN[3]<>(PIN[1] NAND PIN[2]) THEN ERROR(1);
   END;

ERROR(0);
END.
```

In this program the FOR TO DO loop causes the variable to run from 0 to 3. PIN[1] and PIN[2] are assigned the lowest bits from I at every iteration; PIN[1] bit 1 and PIN[2] bit 0. Therefore PIN[1] and PIN[2] assume all possible combinations. The next line checks whether PIN[3] corresponds to the NAND operation of PIN[1] and [2], and if this is not the case (<> !), the statement ERROR(1) is executed. In a way 'ERROR(1)' replaces the WAIT command, only that here the program is terminated and the message 'IC Defect' appears on the screen. The statement 'ERROR(0)' informs the user when an IC is operational: this terminates the program too, but in this case 'IC OK'is displayed.

Declaring input and output pins can prove time-consuming when several gates are involved which is why IC-PASCAL allows these to be defined with a single statement:

```
          PIN[1,2,4,5,9,10,12,13] : INPUT;
          PIN[3,6,8,11] : OUTPUT;
```

If we wish to extend the above program example for the SN 7400 to encompass all four gates then it must look like this:

```
#NAME 7400,SN7400

#PIN 14

#PROGRAM

BEGIN
PIN[1,2,4,5,9,10,12,13] : INPUT;
PIN[3,6,8,11] : OUTPUT;
PIN[7] : GND;
PIN[14] : +5V;

FOR I:=0 TO 3 DO
   BEGIN
   PIN[1] & PIN[2]:=I;
   PIN[4] & PIN[5]:=I;
   PIN[9] & PIN[10]:=I;
   PIN[12] & PIN[13]:=I;
   IF PIN[3]<>(PIN[1] NAND PIN[2]) THEN ERROR(1);
   IF PIN[6]<>(PIN[4] NAND PIN[5]) THEN ERROR(1);
   IF PIN[8]<>(PIN[9] NAND PIN[10]) THEN ERROR(1);
   IF PIN[11]<>(PIN[12] NAND PIN[13]) THEN ERROR(1);
   END;

ERROR(0);
END.
```

When testing OPEN-COLLECTOR components the output pins must be loaded during the query; otherwise the IC will give an incorrect result. For this reason IC-PASCAL incorporates its own load mode.

The load mode is attained with the statement 'LOADMODEON'. For safety reasons IC-PASCAL then immediately prevents a change in the supply voltage, new or redefinition of the input and output pins or the assignment of LOW and HIGH; a pin can only be positively or negatively loaded. The load mode is exited with 'LOADMODEOFF'.

This operating mode is demonstrated below taking the IC SN 7041 (4 x NAND Gate with OPEN-COLLECTOR outputs) as an example:

```
#NAME 7401,SN7401

#PROGRAM

BEGIN
PIN[2,3,5,6,8,9,11,12] : INPUT;
PIN[1,4,10,13] : OUTPUT;
PIN[7] : GND;
PIN[14] : +5V;

FOR I:=0 TO 3 DO
   BEGIN
   PIN[2]&PIN[3]:=I;
   PIN[5]&PIN[6]:=I;
   PIN[8]&PIN[9]:=I;
   PIN[11]&PIN[12]:=I;
   LOADMODEON;
   PIN[1,4,10,13] : LOAD HIGH;
   IF PIN[1]<>(PIN[2] NAND PIN[3]) THEN ERROR(1);
```

```
      IF PIN[4]<>(PIN[5] NAND PIN[6]) THEN ERROR(1);
      IF PIN[10]<>(PIN[8] NAND PIN[9]) THEN ERROR(1);
      IF PIN[13]<>(PIN[11] NAND PIN[12]) THEN ERROR(1);
      LOADMODEOFF;
      END;

   ERROR(0);
   END.
```

As in the test program '7400'the program '7401'firstly defines the input, output and voltage carrying pins. Then I is declared as control variable from 0 to 3; the inputs of of the four gates receive the current value of I every time the loop is repeated. The statement 'LOADMODEON'switches the interpreter to the load mode. Changing the state of the pins from LOW or HIGH is no longer possible. Thereafter, the output pins 1, 4, 10, and 13 are positively loaded (LOAD HIGH) and their state is tested for the NAND operation of the inputs. If these agree the interpreter switches off the load mode with the statement 'LOADMODEOFF'and increments I by 1 (FOR TO DO loop !). As soon as the load mode is exited allocation of 'LOW'or 'HIGH'or 0 or 1 to the input pins is once again allowed.

RC combinations to test mono-flops and components of this kind are allocated in IC-PASCAL using 'RC':

```
PIN[5,6,4] : RC;
...
PIN[5,6,4] : RC OFF;
```

The following seven combinations are possible:

	R	R/C	C
PIN	5	6	4
PIN	+5V	4	3
PIN	+5V	9	8
PIN	+5V	14	13
PIN	+5V	16	14
PIN	+5V	17	16
PIN	+5V	16	17

'+5V' at R means that the respective pin must be the +5V voltage-carrying pin. If the voltage supply has not been assigned at that point an error message appears on the screen.

5. Further Commands in IC-PASCAL

In this section we would like to explain several IC-PASCAL commands which have not been included under the previous headings.

HALT	terminates the execution of the program
PINS:=	allocates a new pin count
IOPORT:=	allocates a new hardware IO address
NOT (...)	negates the following expression, e.g. NOT(PIN[3]) results in 1, when PIN[3] is 0, and 0, when PIN[3] is 1
LO (...)	gives the least significant byte of the expression in brackets

HI (...)	gives the most significant byte of the expression in brackets
CHR (...)	gives the ASCII symbol of the expression in brackets
ORD (...)	gives the number of the expression in brackets
TRUE	corresponds to 'condition fulfilled' (true, see 3.4.2.)
FALSE	corresponds to 'condition unfulfilled' (false, see 3.4.2.)
KEYPRESSED	delivers TRUE, when a key is pressed, otherwise FALSE
READKEY	delivers the key last pressed
WHEREX	gives the current X coordinate of the cursor (column)
WHEREY	gives the current Y coordinate of the cursor (line)
... OR ...	gives the logical OR operation of the preceeding and following expression
... NOR ...	= logical NOR operation
... XOR ...	= logical Exclusive OR operation
... EXOR ...	= logical Exclusive OR operation
... NEXOR ...	= logical NOT Exclusive OR operation
... AND ...	= logical AND operation
... NAND ...	= logical NAND operation
... SHL ...	shifts the preceding expression bit by bit to the left and does so as long as the following expression is large
... SHR ...	shifts the preceding expression bit by bit to the right and does so as long as the following expression is large

•