4. CARP Instructions Reference Guide

%

Syntax

% (operand)

Remarks

The % operator precedes a sequence of ones (1) and zeroes (0) that are interpreted as a bit sequence. Therefore, the operand may only consist of a sequence of 1s and 0s.

By this, you can do bit manipulation in a more explicit form compared to treating integers as bit values.

Example

```
RECIPE  XYSize  =  60;
        Zet     =  4;
        Colors  =  4;

CONST dead      = %00; (* bit sequence 00 *)
      just_died = %01; (* bit sequence 01 *)
      just_born = %10; (* bit sequence 10 *)
      alive     = %11; (* bit sequence 11 *)
              (*                  A  *)
              (*                  |  *)
              (*           "alive bit" *)

REF   east[1,0];
      west      [-1,0];
      north     [0,-1];
      south     [0,1];
      north_ea       [1,-1];
      north_we  [-1,-1];
      south_ea  [1,1];
      south_we  [-1,1];

PROC add_second_bit::(* procedure evaluates second bit of*)
                  (* neighbors that indicates alive   *)
                  (* state and returns sum of found bits*)
BEGIN
RETURN ((east XOR %01) SHR 1) + ((west XOR %01) SHR 1)
  + ((north XOR %01) SHR 1) + ((south XOR %01) SHR 1)
  + ((north_ea XOR %01) SHR 1) + ((north_we XOR %01) SHR 1)
  + ((south_ea XOR %01) SHR 1) + ((south_we XOR %01) SHR 1)
END add_second_bit;
```

*

Syntax

(operand) * (operand)

Remarks

The * operator multiplies two operands. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;

EVENT 1;
b:= 17;
a := b * 4;
```

+

Syntax

(operand) + (operand)

Remarks

The + operator adds two operands. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
    VAR temp;
    CONST c = 2345;

EVENT 1;
temp:= c + 37;
```

-

Syntax

(operand) - (operand)

Remarks

The - operator subtracts two operands. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;

EVENT 1;
b:= 17;
a := b - 4;
```

:=

Syntax

(VAR) identifier | Self := expression;

Remarks

The assignment procedure ':=' attributes the value of an expression right to the assignment procedure to any variable or the cell Self standing left to this procedure.

Example

```
EVENT E1;
PARALLEL DO
  Self := OddCell;
OD;
ShowPlane;
```

<

Syntax

(operand) < (operand)

Remarks

The < operator compares two operands in respect to size. If the first operand is smaller than the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
WHILE b < i DO
a := b + 48
OD;
```

<=

Syntax

(operand) <= (operand)

Remarks

The <= operator compares two operands in respect to size. If the first operand is smaller or equal than the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
IF b <= limit
THEN b := Any8Sum (a, b, c, d, e, f, g, h);
FI;
```

<>

Syntax

(operand) <> (operand)

Remarks

The <> operator compares two operands in respect to unequality. If the first operand is unequal to the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
IF a <> limit
THEN a := a +1
ELSE a := limit
FI;
```

=

Syntax

(operand) = (operand)

Remarks

The = operator compares two operands in respect to equality. If the first operand is equal to the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
WHILE a = b DO
 Self := add_four_positions;
OD;
```

>

Syntax

(operand) > (operand)

Remarks

The > operator compares two operands in respect to size. If the first operand is greater than the second, the whole expression returns true, otherwise false.

Example

```
VAR a, b;

EVENT 1;
IF b > a
  THEN b := a;
FI;
```

>=

Syntax

(operand) >= (operand)

Remarks

The >= operator compares two operands in respect to size. If the first operand is greater or equal compared to the second, the whole expression returns true, otherwise false..

Example

```
VAR a, b;

EVENT 1;
WHILE a >= 0 DO
  a := a -1;
OD;
```

A

AND

Syntax

(operand) AND (operand)

Remarks

The AND operator connects two operands and returns true, if both operands are true. All other cases return false.

If the cells of your CAT model may only have the state 0 or 1, you may treat cells with the AND operator, too.

Example

```
a := 8;
IF (a > limit) AND (b = 9)
  THEN a := Self FI;
```

Any8Sum

Syntax

Any8Sum (n1, n2, n3, n4, n5, n6, n7, n8);

Remarks

Any8Sum adds the state values of neighbors, variables or constants that follow as 8 parameters.

Example

```
REF   knight_t_l     [-1,-2]; (* possible jumps of *)
      knight_t_r     [1,-2];  (* knights *)
      knight_b_l     [-1,2];
      knight_b_r     [1,2];
      knight_mt_l    [-2,-1];
      knight_mt_r    [2,-1];
      knight_mb_l    [-2,1];
      knight_mb_r    [2,1];

EVENT E1;
PARALLEL DO
                Self := Any8Sum
                (knight_t_l, knight_t_r, knight_b_l, knight_b_r,
                knight_mt_l, knight_mt_r, knight_mb_l, knight_mb_r);
OD;
ShowPlane;
```

B

BarrelForm

Syntax

BarrelForm;

Remarks

The topology BarrelForm forms a virtually barrelshaped matrix, i.e. the right and left edges of the cell matrix are mutually copied to the opposite edge.

```
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
```

Topology BarrelForm (c = copied cell)

Example

```
                EVENT SetUp;
```

```
        BarrelForm;
        PlClipActive;
        ShowPlane;
```

Beep

Syntax

```
Beep ( n );
```

Remarks

Returns n beeps.

This procedure is useful if you want to mark a crucial state of your cellular automaton model by an acoustic signal.

Example

```
CONST max_value = 5478;
VAR x;
EVENT E0;
...
IF x >= max_value THEN Beep(1) FI;
```

BEGIN ... END

Syntax

```
BEGIN
  statement;
  [statement;]
  ...
  [statement;]
END;
```

Remarks

Instructions bracketed by the keywords BEGIN and END may be used as an additional means for structuring a CARP program. Usage is optional.

Example

```
(* Compound statement used within an "IF" statement *)
IF First < Last THEN
BEGIN
  Temp := First;
  First := Last;
  Last := Temp;
END;
```

```
FI;
```

Brake

Syntax

Brake;

Remarks

Stops the current event.

Example

```
FOR x := 1 TO x < 10 BY 2 DO
        WRITE (x);
        IF (x + 3) = 5
              THEN Brake
              ELSE y := x + 1
        FI;
OD x;
```

C

Comments in a CARP program

Syntax

(* string *)

Remarks

To keep your program self-explanatory even for later times, use comments in your CARP program. Use pairs of "(*" and "*)" respectively to indicate start or end of a comment. Comments may comprise several lines.

Example

```
REF left[-1,0]; up[0,-1]; right[0,1];
(* x counts negative for referenced cells
   on the top of cell Self *)
```

Colors

Syntax

```
Colors = n;
```

Remarks

Colors defines the number of available colors. The color actually assigned to a certain state may be either interactively set by means of the color customizing button or by means of the RGBBrush procedure.

Example

```
RECIPE  XYSize  =  140;
        XYBound = 3 ;
        Zet = 20;
        Colors = 20;
```

CONST

Syntax

```
CONST
  identifier = expression;
```

Remarks

A constant declaration (CONST) defines an identifier, which denotes a constant value within the block containing the declaration. A constant identifier cannot be included in its own declaration. You can only assign a value to a constant during the declaration.

Expressions used in constant declarations must be written in such a way that the compiler can evaluate them at compile time.

A string cannot be assigned to a constant. If possible, use instead the WRITE procedure with a string parameter.

Examples

```
(* Constant Declarations *)

CONST
  limit  = 65000;
  KeyCode  = 943762;
```

D

DelBrushes

Syntax

```
DelBrushes;
```

Remarks

Deletes all color palette entries, which may be defined by means of the RGBBrush or the RGBPalette procedure. Be careful! The color palette has to be redefined after the entries have been deleted by the DelBrushes procedure.

Deleting of color palette entries may also affect the color display of MS Windows or other Windows applications.

Example

```
EVENT SetUp;
DelBrushes; (* all color palette entries
                        are now lost *)
RGBPalette (10, 0,20, 0,20, 0,20);
        (* color palette is now redefined *)
```

DIV

Syntax

```
(operand) DIV (operand)
```

Remarks

The DIV operator devides two operands and returns an integer as result of the whole expression. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;
EVENT 1;
b:= 17;
a := b DIV 4; (* a is now 4 *)
```

E

EVENT

Syntax

```
EVENT [E<n>] | [SetUp];
            statements;
[END.] | [EVENT E <n+1>;]
```

Remarks

An event is a program part that starts with the keyword "EVENT" plus identifier number plus semicolon and ends with the next keyword "EVENT" or the keyword "END.". The identifier number must be in the range from 0 to 5. An event is a program unit, which may be triggered by its

corresponding single step or run button or by the SetUp button and whose code can be executed independently at a time.
Each event should have one dominant function, e.g. initialization or the implementation of a specific algorithm.

One CARP program may contain up to 6 events with identifiers from "E0" to "E5" and, additionally, the special event SetUp.

Example

```
EVENT E0; (* initialization of cell plane *)
PlFillRandom (Dead,Alive);
ShowPlane;

EVENT E1;
```

Expressions

Expressions consist of operators and operands. These are the operands:

constants
A constant declaration (CONST) defines an identifier, which denotes a constant value within the block containing the declaration. A constant identifier cannot be included in its own declaration.

variables
A variable (VAR) declaration associates an identifier and a type with a location in the memory where values of that type can be stored.

procedures
A procedure may be either predefined or user-defined. User-defined procedures may be function procedures or procedures using a side effect.

operators
The different types of operators existing in CAT (arithmetic operators, logic operators, comparative operators, bit operators) allow to join operands.

Subexpressions can be enclosed in parentheses to change the order of precedence.

F

FOR ... TO ...BY ... DO ... OD

Syntax

```
FOR assignment TO expression [BY step] DO
            statement;
OD loop_variable;
```

Remarks

The FOR ... OD instruction causes the statement after DO to be executed once for each case the Boolean expression is true. The Boolean expression is checked **after** the first execution of statement sequence. So, statement sequence is executed at least one time.

The loop variable is implicidly defined and may not be defined at the top of your CARP program. The loop variable may be read inside a loop, but never be written to. After the loop is completed the content of the loop variable is not defined any more.

If the BY construct is used, you can change the interval by which the loop variable is incremented to the value which follows BY.

Example

```
FOR x := 1 TO x < 10 BY 2 DO
     WRITE (x);
     IF (x + 3) = 5
       THEN Brake
       ELSE y := x + 1
     FI;
OD x;
```

G

GetX

Syntax

GetX;

Remarks

Returns the current x-value of the treated cell inside a PARALLEL DO loop.

Example

```
EVENT E1;
   PARALLEL DO
     ...
     IF top > 0
     THEN
         WRITE ('','Current x value : ', GetX);
     FI;
   OD;
```

GetY

Syntax

GetY;

Remarks

Returns the current y-value of the treated cell inside a PARALLEL DO loop.

Example

```
EVENT E1;
   PARALLEL DO
     IF top = 1
     THEN
                    WRITE (GetY);
     FI;
   OD;
   ShowPlane;
```

I

Identifiers

Identifiers denote the following:

CONST(ants)
PROC(edures programs)
VAR(iables)

Identifiers can be formed of up to 31 characters.

- The first character of an identifier must be a letter. Upper or lower case letters are allowed at any place.
- The characters that follow the first one must be letters, digits, or underscores (no spaces).

Like reserved words, identifiers are **case-sensitive**. Identifier may not coincide with reserved words.

Examples

```
(* Identifiers *)
VAR Limit;
CONST A_State = 4;
     B_State = 8;
```

IF .. THEN .. ELSE .. FI

Syntax

IF expression THEN statement [ELSE statement] FI;

Remarks

IF, THEN and ELSE specify the conditions under which a statement will be executed.

If the Boolean expression after IF is true, the statement after THEN is executed. Otherwise, if the ELSE part is present, the statement after ELSE is executed.

Example

```
x := Random (1000);
IF  (x > 995)
            THEN Self := Alive;
            ELSE Self := Dead;
FI;
```

INV

Syntax

INV (operand)

Remarks

The INV operator has an integer or bit operand and converts all its zeroes to ones and all ones to zeroes. As operand is allowed: an integer constant, a variable, a procedure that returns an integer or a bit operand.

Example

```
VAR a, b;

EVENT E4;
a := %110; (* 6 *)
b := INV a;
WRITE ('', ' a: ', a);
WRITE ('', ' INV a: ', b); (* result : - 7 *)
END.
```

M

MOD

Syntax

(operand) MOD (operand)

Remarks

The MOD operator devides two operands and returns the remainder as the result. As operands are allowed: integer constants, variables or procedures that return an integer.

Example

```
VAR a, b;

EVENT 1;
b := 17;
a := b MOD 4;
```

MooreSum

Syntax

MooreSum

Remarks

MooreSum adds the state values of the northern, southern, western, eastern, northeastern, northwestern, southeastern and southwestern neighbors of Self.

m m m m m

```
m O O O m
m O S O m
m O O O m
m m m m m
```

MooreSum (O = evaluated cell)

Example

```
EVENT E1;
PARALLEL DO
            IF (MooreSum <> 4)
              Self := Ill;
            FI;
OD;
ShowPlane;
```

N

NeumannSum

Syntax

NeumannSum

Remarks

NeumannSum adds the state values of the northern, southern, western and eastern neighbors of Self.

```
m m m m m
m m O m m
m O S O m
m m O m m
m m m m m
```

NeumannSum (O = evaluated cell)

Example

```
EVENT E1;
PARALLEL DO
            IF (NeumannSum > 4)
            Self := Red;
            FI;
OD;
ShowPlane;
```

NOT

Syntax

NOT (operand)

Remarks

The NOT operator negates the result of the Boolean expression or operand that follows.

If the cells of your CAT model may only have the state 0 or 1, you may also use a cell denoter as operand for NOT.

Example

```
IF NOT (a > limit)
  THEN a := Self
FI;
```

O

OddCell

Syntax

OddCell

Remarks

OddCell provides access only to those cells whose x-value in the matrix is odd. This effects a cell matrix resembling a chess-board.

The x-value is counted from the first top left cell to the last right bottom cell continuously. That means for example that for a matrix with XYSize 31 the first cell of the second row is considered even (i.e. 32nd cell).

Example

```
EVENT E1;
PARALLEL DO
            Self := OddCell;
OD;
ShowPlane;
```

OR

Syntax

(operand) OR (operand)

Remarks

The OR operator connects two operands and returns true, if one or both operands are true. The remaining case returns false.

If the cells of your CAT model may only have the state 0 or 1, you may treat cells with the OR operator, too.

Example

```
IF (a > limit) OR (Self = 9)  THEN a := Self FI;
```

P

PARALLEL DO

```
PARALLEL DO
  statement;
  [statement;]
OD;
```

Remarks

The PARALLEL DO executes the instructions of its body once for all cells of the cell matrix in parallel.

Internally, a copy of the present state of all cells at the beginning of the PARALLEL DO construct is made, so that all conditional instructions etc. take the value contained in this copy. At the end, the computed state of all cells is written back and kept for future evaluations.

Mostly, the Self procedure is used inside a PARALLEL DO construct as an important part of a cell-related algorithm.

Example

```
EVENT E1;
    PlClipActive;
    PARALLEL DO
      Self := North XOR South XOR East XOR West;
    OD;
    ShowPlane;
```

ParallelMethod

Syntax

```
ParallelMethod;
```

Remarks

Is now the default method and needs not to be particularly defined.

In a future version of CAT, there will also be a method SequentialMethod.

Example

- - -

PillowForm

Syntax

PillowForm;

Remarks

The topology PillowForm assumes an axis in the middle of the matrix. Cells of the edges that have the same distance to this axis are copied to their counterpart.

```
c4c3c2c1|c1c2c3c4
c m m m | m m m c
c m m m | m m m c
c m m m | m m m c
c m m m | m m m c
c m m m | m m m c
c4c3c2c1|c1c2c3c4
```

Topology PillowForm (c = copied cell)

Example

EVENT SetUp;
PillowForm;
PlClipActive;
ShowPlane;

PipeForm

Syntax

PipeForm;

Remarks

The topology PipeForm forms a virtually pipe-shaped matrix (tube), i.e. the top and bottom edges of the cell matrix are mutually copied to their opposite edges.

```
c c c c c c c
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
c c c c c c c
```

Topology PipeForm (c = copied cell)

Example

EVENT SetUp;
PipeForm;
PlClipActive;

```
ShowPlane;
```

PlClipActive

Syntax

```
PlClipActive;
```

Remarks

Restricts the effect of the subsequent instructions to the cell matrix without its border areas defined by an optional XYBound declaration. This setting is the default value.

Example

```
EVENT SetUp;
  PlClipActive;
  RGBPalette (2, 20,10, 60,10, 80,10);
```

PlClipAll

Syntax

```
PlClipAll;
```

Remarks

Makes the whole cell matrix including the border areas available for subsequent instructions This procedure is only useful if you want to initialize the border areas.

Example

```
EVENT SetUp;
  PlClipAll;
  RGBPalette (2, 20,10, 60,10, 80,10);
```

PlClipXY

Syntax

```
PlClipXY ( x, y );
```

Remarks

Shows a central portion of the whole cell matrix with the corresponding x- and y-values. Thus, you can focus an area of special interest.

Example

```
EVENT E4;
   ...
   PlClipXY (10,10);
```

PlFillRandom

Syntax

PlFillRandom (Low, High);

Remarks

The procedure PlFillRandom initializes the cell matrix by random values ranging from parameter Low to High.

Keep in mind the range of states, which are defined by the Zet declaration. If the range of possible values produced by PlFillRandom exceeds the number of defined states, the range is restricted to the Zet value.

To take effect the parameters must be inside the range of defined colors and states (cp. Zet) and the smaller value **must** precede the greater one.

Example

```
EVENT E0;
PlFillRandom (0,10);
ShowPlane;
```

PlFillUni

Syntax

PlFillUni (n);

Remarks

Gives the whole cell matrix a uniform state, which is defined by the n parameter, and via the associated color mapping value a uniform appearance.

To take effect the parameter must be inside the range of defined colors and states (cp. Zet).

Example

```
EVENT SetUp;
  PlClipActive;
  PlFillUni (32);
  ShowPlane;
```

PlFillUpStairs

Syntax

PlFillUpStairs (Low, High, By);

Remarks

The procedure PlFillUpStairs creates a stair-like shape in the cell matrix. Thereby, the parameter Low gives the lower state and colormapping value , High the higher state and colormapping value and By the interval in which the range between Hi and Lo is filled. Useful for initialization purposes.

To take effect the parameters must be inside the range of defined colors and states (cp. Zet, Colors).

Example

```
EVENT SetUp;
  PlClipAll;
  PlFillUpStairs (2, 20, 4);
  ShowPlane;
```

PROC

Syntax

```
PROC proc_identifier [VAR (identifier, identifier...)] :;
[VAR identifier;]
[CONST identifier;]
BEGIN
            statement sequence;
[RETURN expression;]
END proc_identifier;
```

Remarks

A procedure is a program part, which performs a specific action, often based on a set of parameters. CAT provides both the function procedure that returns a value and the normal prodecure that exchanges data with the CARP program via variables declared in the procedure head.

The procedure heading specifies the identifier for the procedure and the formal parameters (if any). A procedure is activated by a procedure call.

The procedure heading is followed by:

- a declaration part that declares local objects,
- the statements between BEGIN and END, which specify what is to be executed when the procedure is called.

A function procedure contains the keyword RETURN followed by an expression as last instruction.

Example

```
REF   knight_t_l     [-1,-2];
      knight_b_r     [1,2];
      knight_mt_l    [-2,-1];
      knight_mb_r    [2,1];

(* procedure adds four positions that might be reached by knight moves *)
PROC add_4_positions (VAR ret):;

BEGIN
ret := knight_t_l + knight_b_r + knight_mt_l + knight_mb_r;
END add_4_positions;

PROC add_4_pos:; (* the same more briefly and the        *)
BEGIN            (* procedure returning the value itself *)
RETURN knight_t_l + knight_b_r + knight_mt_l + knight_mb_r;
END add_4_pos;

EVENT E3;
                PARALLEL DO
                  WRITE ('', 'Value : ', add_4_pos);
     OD;
     ShowPlane;
```

R

Random

Syntax

Random (n);

Remarks

Returns a random number between 0 and n. Negative n values are not allowed.

Every call of a loop that contains the Random instruction produces the same sequence of results for internal reasons. If you want to avoid this effect, use Randomize additionally.

Example

```
VAR x;

EVENT E0;
...
x := Random (1000);
IF x > 950 THEN add_four_positions FI;
```

Randomize

Syntax

Randomize;

Remarks

Creates a new base number for the random number generator.

This procedure is advisable if you want to prevent that each loop (PARALLEL DO, WHILE), that contains a Random procedure produces the same sequence of random numbers. The Randomize procedure should be used in the event SetUp or in the event containing the Random procedure.

Randomize should **not** be used, if you are searching for a program error that is related to random numbers.

Example

```
EVENT SetUp;
   RGBPalette(Colors, $0, 10, $32,10, $B6,10,);
   Randomize;
```

RECIPE

Syntax

```
RECIPE
[XYSize and/or XYBound declarations;]
[VAR declarations;]
[CONST declarations;]
[REF declarations;]
[PROC declarations;]
EVENT declarations;
statements;
END.
```

Remarks

A CARP program has to be started by the keyword "RECIPE" and terminates with the keyword "END." ("END" followed by a point). Between these delimiters, you can declare variables, constants, refered neighbors of a cell, user defined procedures and - as independently executable parts of a CARP program - events.

Normally, the keyword RECIPE is followed by settings for the size of the cell matrix and evaluated neighborhood, by definitions of constants (CONST) , variables (VAR) or referenced cells (REF) and by events (EVENT) that contain the proper program functions. A template for a program may look as follows:

Example

```
RECIPE  XYSize  =  50;
CONST ...;
VAR ...;
REF ...;

EVENT SetUp;
...

EVENT E0;
...

EVENT E1;...
...
END.
```

REF

tt.01.jj

Syntax

REF identifier [xvalue,yvalue];   *(read)*

Remarks

The REF declaration assigns a name to specified neighboring cells of the cell Self and allows such to refer to the value of these identified cells by their name. Precondition: The cell referred to may not exceed the limits set by XYBound.

To use the value of a certain reference cell you have to do two things:
- Define a referenced cell.
- Use the defined neighbors within the program by referring to their names. Compare the sample program part on the bottom:

Note:
- You may only read from referenced cells, not write to them. This is restricted to the procedure Self.
- X-values to the right of Self and Y-values on the bottom of Self have a positive value.

Example

```
REF   right_neighbor         [1,0];
      left_neighbor          [-1,0];
      top_neighbor           [0,-1];
      bottom_neighbor        [0,1];
...

EVENT E1;
    PARALLEL DO
      Self := top_neighbor OR left_neighbor OR Self OR
              right_neighbor OR bottom_neighbor;
    OD;
    ShowPlane;
END.
```

RePaint

Syntax

RePaint;

Remarks

Paints the graphic window again, if appearance or colors of the cell matrix are garbled. Scarcely useful inside a CARP program, compare instead the corresponding RePaint button.

Example

- - -

REPEAT .. UNTIL

Syntax

```
REPEAT
  statement;
  [statement;]
UNTIL expression;
```

Remarks

The statements between REPEAT and UNTIL are executed in sequence until, at the end of the loop body, the Boolean expression after UNTIL is true.

The sequence is executed at least once. The delimiter of the REPEAT ... UNTIL loop is a ';'.

Example

```
x := 1;
REPEAT
        IF (x + 3) = 8
             THEN WRITE (x);
                x := x + 1
             ELSE x := x + 1
        FI;
UNTIL x > 15;
```

RGBBrush

Syntax

```
RGBBrush ( n, r, g, b );
```

Remarks

Assigns the color mapping n the colors given by the parameters r(ed), g(reen) and b(blue).

This procedure is advisable, if you want to assign certain cell states to specific colors. (sample a)

This procedure may also be used if you want to change the previous overall color settings for a special color at a given time (sample b).

Example

(sample a)

```
RECIPE  XYSize  =  60;
    Zet     =  4;
    Colors  =  4;

CONST dead      = %00; (* bit sample 0000 *)
   just_died = %01; (* bit sample 0001 *)
   just_born = %10; (* bit sample 0010 *)
   alive     = %11; (* bit sample 0011 *)
EVENT E0;
   RGBBrush (dead, 0, 0, 0);          (* black *)
   RGBBrush (just_died, 152, 88, 46); (* brown *)
   RGBBrush (just_born, 74, 229, 3);  (* light green *)
   RGBBrush (alive, 50, 174, 30);     (* dark green *)
```

(sample b)

```
VAR cell_state;

EVENT E4;
IF generation_counter > 100
        THEN RGBBrush (cell_state, 24 ,30, 30)
FI;
```

RGBPalette

Syntax

RGBPalette (n, r0, ri, g0, gi, b0, bi);

Remarks

The procedure RGBPalette allows to define a set of colors and their dissemination on the color palette. These parameters have to be defined:

n	Number of colors to define. Generally, this number should comply with the number of defined states (Zet).
r0	Starting point for the red value.
ri	Increment value by which the red value increases. Values above 255 are corrrected to a maximum value 255.
g0	Starting point for the green value.
gi	Increment value by which the green value increases. Values above 255 are corrrected to a maximum value 255.
b0	Starting point for the blue value.
bi	Increment value by which the blue value increases. Values above 255 are corrrected to a maximum value 255.

Some general remarks: each defined color is a set of three values for their portion of red, green and blue (rgb). Each of this component color has a definition range from 0 to 255 (hexadecimal $0 to $FF). Red, green and blue each set to 255 result in the color white, red, green and blue each set to 0 result in the color black. That is the area, in which you may select certain colors.

Note:
RGBPalette sets the colors for your automaton tool model in a global way. Besides, you may define a specific color by means of the RGBBrush procedure.
Colors defined either by RGBPalette or RGBBrush may be varied interactively later on by means of the color customizing button. To use this button for particular colors is most advisable, because it is very difficult to predict the resulting color only by defining the red, green and blue parameters.
Values for increments (ri, gi, bi) may also be negative. This makes sense together with high starting values for r0, g0 or b0.
Values may be given as decimal or hexadecimal figures with leading $.

The example program below will generate this color palette:

		r-value	g-value	b-value
color 1	30	40	50	
color 2	45	55	65	
color 3	60	70	80	
color 4	75	85	95	
color 5	90	100	110	

Example

```
EVENT SetUp;
RGBPalette (5, 30,15, 40,15, 50,15);
```

RingForm

Syntax

RingForm;

Remarks

The topology RingForm forms a virtual endless matrix connecting at first two edges and then the edges of the built up pipe. This body is also known as thorus. The RingForm topology is the **default setting** and therefore, the RingForm instruction may be omitted.

```
c c c c c c c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c m m m m m c
c c c c c c c
```

Topology RingForm (c = copied cell)

Example

EVENT SetUp;
RingForm;
PlClipActive;
ShowPlane;

S

Self

Syntax

Self *(read / write)*

Remarks

The only instruction to change the state of a cell and thereby the whole cell matrix is Self. All other cell matrix-related procedures only allow reading of a cell state.

Self is strongly connected with the PARALLEL DO instruction. Inside a PARALLEL DO cycle, Self allows for each cell read or write access.

All instructions inside a PARALLEL DO and related to Self and other referred cells have to be thought of as actually happening **simultaneously**. (In fact, on a single CPU computer, a copy of the state of all cells will be made, and, depending on these values, the instructions for all cells will be of course carried out subsequently.) But focussing on CAT's concept, Self and PARALLEL DO are the decisive keys to leave array treatment and such things behind and turn to the new programming paradigm 'the cell in its environment'.

The effect of the sample instructions below (it implements Conveys Life program): For each cell of the cell matrix will be controlled as to whether Self is 'alive' (read access) and has two or three 'alive' neighbors ('alive' is assigned to the state 1 of a cell). If this is true, Self will be set to 'alive' (write access with the ':=' procedure). Otherwise, if Self is 'dead' and has three 'alive' neighbors, Self will be set again to 'alive'. In all other cases, Self will be considered as 'too lonely' or 'overcrowded' and therefore set to 'dead'.

Example

```
PARALLEL DO
            IF (Self = Alive) AND
                ((MooreSum = 3) OR (MooreSum = 2))
            THEN Self := Alive
            ELSE IF (Self = Dead) AND (MooreSum = 3)
                    THEN Self := Alive
                    ELSE Self := Dead
                FI
            FI;
OD;
```

SetLattice

Syntax

SetLattice (thickness, foregroundcolor, backgroundcolor);

Remarks

Returns a lattice pattern originating from the center of your cell matrix with free spaces of size 'thickness' and with the corresponding fore- and backgroundcolors.

Example

```
EVENT SetUp;
PlClipActive;
SetLattice(3,1,19);
```

SheetForm

Syntax

SheetForm;

Remarks

The topology SheetForm makes the evaluation of algorithms end on the edges of the cell matrix without any further continuation on other edges.

m m m m m m m

```
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
m m m m m m m
```

Topology SheetForm (m = normal cell, no copied cell)

Example

```
EVENT SetUp;
SheetForm;
PlClipActive;
ShowPlane;
```

SHL

Syntax

(operand) SHL (operand)

Remarks

The SHL operator shifts all bits of a binary digit by the value of the second operand times to the left. Leading digits are filled by 0.

This works for integer variables, constants or referenced cells interpreted as binary values as well as for explicitly defined binary digits. In the following sample the variable b returns the value 12 both times.

Example

```
a := %110; (* 6 *) (* using bit operator *)
b := a SHL 1;
WRITE ('',b);
                                (* without bit operator *)
a := 6;
b := a SHL 1;
WRITE ('',b);
```

ShowCell

Syntax

ShowCell (n);

Remarks

This procedure shows the current state of the cell with the x-value n.

This value is computed as n = x + (XSize * (y - 1)). (An example: in a cell matrix with XYSize = 10 the first cell in the top left corner counts 0 and the last cell in the bottom right corner counts 99.)

This procedure is very CPU-time-consuming and should only be used if the focus is on a single cell.

Example

```
EVENT E1;
    PARALLEL DO
     IF x > delimiter
     THEN
        Self := (NeumannSum + Self) > 0
     FI;
    OD;
    ShowCell (74);
    ShowCell (75);
    ShowCell (76);
```

ShowKind

Syntax

ShowKind (w);

Remarks

Shows the state and color mapping of a single cell.

Useful only if the focus is on these settings of a single cell. Can then be combined with the ShowCell procedure.

Example

```
EVENT E1;
   ShowKind (74);
   ShowKind (75);
   ShowKind (76);
   ShowCell (74);
   ShowCell (75);
   ShowCell (76);
```

ShowPlane

Syntax

ShowPlane;

Remarks

This function is necessary for showing the whole cell matrix in its current state. Should normally occur at the end of any event description for control purposes. If your cellular automaton model is very CPU time-consuming, you can order to display only every tenth or whatever generation of your CAT model.

Never use ShowPlane inside a PARALLEL DO instruction, for this might crash CAT.

Example

```
EVENT E1;
...
IF i < 50
          THEN ShowPlane
          ELSE IF i MOD 10 = 0
                 THEN ShowPlane
                 FI
FI;
i := i +1;
```

SHR

Syntax

(operand) SHR (operand)

Remarks

The SHR operator shifts all bits of a binary digit by the value of the second operand times to the right. Leading digits are filled by 0.

This works for integer variables, constants or referenced cells interpreted as binary values as well as for explicitly defined binary digits.

Example

```
(* life model (cp. Convey) with four different states: *)
(* life, just_born, dead, just_died *)

RECIPE XYBound = 60;

CONST dead      = %00; (* bit sample 0000 *)
      just_died = %01; (* bit sample 0001 *)
      just_born = %10; (* bit sample 0010 *)
      alive     = %11; (* bit sample 0011 *)
            (*                  A  *)
            (*             "alive bit" *)

REF   right_n      [1,0];
      left_n    [-1,0];
      top_n     [0,-1];
      bot_n     [0,1];

PROC add_second_bit:;
(* procedure evaluates second bit of neighbors that *)
(* indicate alive state and returns sum of found bits*)

BEGIN                       *)
RETURN ((right_n XOR %01) SHR 1) + ((left_n XOR %01) SHR 1) + ((top_n XOR %01) SHR 1)   + ((bot_n
XOR %01) SHR 1)
END add_second_bit;

EVENT E0;
      RGBBrush (dead, 0, 0, 0);          (* black *)
      RGBBrush (just_died, 152, 88, 46); (* brown *)
      RGBBrush (just_born, 74, 229, 3);  (* light green *)
      RGBBrush (alive, 50, 174, 30);     (* dark green *)

EVENT E1;
PARALLEL DO
a := add_second_bit;

IF (a = 2) OR (a = 3)
  THEN IF (a = 3) AND ((Self = dead) OR (Self = just_died))
       THEN Self := just_born;
       ELSE Self := alive
       FI;
  ELSE IF (Self = alive) OR (Self = just_born)
       THEN Self := just_died;
       ELSE Self := dead;
```

```
    FI;
FI;
OD;
ShowPlane;

END.
```

Statement

A statement is one of the following:

assignment (:=)
BEGIN..END
FOR..TO..BY..DO..OD
PARALLEL...DO
IF..THEN..ELSE FI
PROC(edure)
REPEAT .. UNTIL
WHILE .. DO .. OD

V

VAR

Syntax

```
  VAR
    identifier, ... identifier;
```

Remarks

A variable (VAR) declaration associates an identifier with a location in the memory where values can be stored.

You may not combine a declaration of a variable with an assignment like you might expect from the usage of constants. Assign a value to the variable inside an event.

Examples

```
(* Variable Declarations *)
  VAR
    x , y , z;

x := 3;
```

W

WHILE ... DO ... OD

Syntax

WHILE expression DO statement OD;

Remarks

A WHILE statement contains an expression, which controls the repeated execution of one or several statements embraced by the keywords 'DO' and 'OD'. The statement after DO is executed repeatedly as long as the Boolean expression is true.

The expression is evaluated before the statement is executed, so if the expression is false at the beginning, the statement will not be executed at all.

Example

```
WHILE i < 20 DO
            Self := NeumannSum DIV 2;
            i := i + 1 OD;
```

WinClipActive

Syntax

WinClipActive;

Remarks

Shows only the active part of the cell matrix without any border areas. WinClipAll or WinClipActive are only relevant for the **appearance** of the STATE window.
The same can be achieved interactively by means of the magnifier button.

Example

```
EVENT SetUp;
   RGBPalette(Colors, $0, $FF, $32,0, $B6,0);
   WinClipActive;
   ShowPlane;
```

WinClipXY

Syntax

WinClipXY (x, y);

Remarks

Shows a central portion of the whole cell matrix with the corresponding x- and y-values. Thus, you can focus on an area of special interest.

Example

```
EVENT E4;
PlFillRandom (1,4);
WinClipXY ( 5, 5);
```

WinClipAll

Syntax

WinClipAll;

Remarks

Shows the whole cell matrix including the border areas. WinClipAll or WinClipActive are only relevant for the **appearance** of the STATE window.

The same can be done interactively by means of the magnifier button.

Example

```
EVENT SetUp;
   RGBPalette(Colors, $0, $FF, $32,0, $B6,0);
   WinClipAll;
   ShowPlane;
```

WrDCaps

Syntax

WrDCaps;

Remarks

Acronym for 'Write Display Capabilities'. The corresponding values specific to your data display are written into the LIST window. Useful for system administration and service purposes, especially on graphic resolution issues. A possible output in the LIST window:

```
 Display capabilities

 H/V Resolution :    1024     768
 Pixel/Planes   :       8       1
 Colors         :      20
 Palette/reserv :     256      20
```

Example

```
EVENT SetUp;
  RGBPalette(Colors, 127, 2, 127,30, 127,30);
  ShowPlane;
  WrDCaps;
```

WRITE

Syntax

WRITE ([string] | [(VAR) identifier] | [(CONST) identifier] | [(PROC) identifier] [:n] ['']);

Remarks

Writes the contents of variables, constants, values of function procedures or strings to the LIST window and the .CAL file. Different operands have to be devided by a comma, strings must be included by ' (apostrophe).

Several facilities are provided for formatting the output:

[(var):n]	If the contents of the variable or the constant has less than n digits, the output is indended accordingly.
' '	Put at the end or beginning of the parameter list, a carriage return / linefeed (CR/LF) at the end or beginning of the output is caused.

If the buffer to which all data is moved is full, you will get the message "Editor buffer is full" and will be prompted whether you want to overwrite the contents or stop writing. All written informations may later be inspected by means of the LIST window.

Example

```
IF  (x > limit)
   THEN WRITE ('','Limit exceeded with value : ');
        WRITE (x : 8, '');
FI;
```

WrMCaps

Syntax

WrMCaps;

Remarks

Acronym for 'Write Memory Capabilities'. The corresponding values of RAM (Random Access Memory) usage specific to your hardware and operating system configuration are written into the LIST window. Useful for system administration and service purposes, especially if you are in doubt about sufficient memory (RAM).

A possible output in the LIST window may look as follows:

```
Memory and resources

Mem_free    KB :   47730
Mem_block   KB :   16320
Sys_Res     %  :      62
GDI_Res     %  :      62
Usr_Res     %  :      83
```

Example

```
EVENT SetUp;
   RGBPalette(Colors, 127, 2, 127,30, 127,30);
   ShowPlane;
   WrMCaps;
```

WrPPars

Syntax

WrPPars;

Remarks

Acronym for 'Write Plane Parameters'. The corresponding values for your actual CAT cell matrix configuration are written into the LIST Window. Useful for system administration and debugging purposes.

A possible output in the LIST window may look as follows:

```
CAT actual parameters

X/YSize   :     31     31
X/YBound  :      3      3
X/YTotal  :     37     37
Act/TotSz :    961   1369
Org/Skip  :    114      6
```

Example

```
EVENT SetUp;
   RGBPalette(Colors, 127, 2, 127,30, 127,30);
   ShowPlane;
   WrPPars;
```

X

XOR

Syntax

(operand) XOR (operand)

Remarks

The XOR operator adds two operands and returns true if one of the two operands returns true and the other false. If both the operands return true or false, the whole expression returns false. As operands are allowed: integer constants, variables, referenced cells or procedures that return an integer.

Example

```
b := 8;
IF (a > limit) XOR (b = 8)
  THEN a := Self
FI;
```

XYBound

Syntax

XYBound = n;

Remarks

Defines the range of the neighborhood of the cell Self (in x and y values) that can be evaluated by any instruction of your CARP program. Referenced cells (REF) must be inside the range of the XYBound.

XYBound defines moreover the width of the border area of the cell matrix that is shown if the PlClipAll procedure is used or that is effected by an equivalent setting of the magnifier button.

Example

```
RECIPE  XYSize  =  140;
        XYBound = 1 ;
```

```
REF   east[1,0];
      west      [-1,0];
      north     [0,-1];
      south     [0,1];
      north_ea        [1,-1];
      north_we  [-1,-1];
      south_ea  [1,1];
      south_we  [-1,1];
```

```
PROC add_second_bit:;(* procedure evaluates second bit of*)
                    (* neighbors indicating alive state *)
BEGIN               (* and returns sum of found bits    *)
RETURN ((east XOR %01) SHR 1) + ((west XOR %01) SHR 1) +
   ((north XOR %01) SHR 1) + ((south XOR %01) SHR 1) +
   ((north_ea XOR %01) SHR 1) +((north_we XOR %01) SHR 1) +
   ((south_ea XOR %01) SHR 1) + ((south_we XOR %01) SHR 1)
END add_second_bit;
```

XYSize

Syntax

XYSize = n;

Remarks

Defines the horizontal (x) and vertical (y) size of a cell matrix. If you want to define a different YSize compared to XSize, you can use the YSizedeclaration.

Keep in mind that high XYSize values are very CPU time-consuming.

Example

```
RECIPE  XYSize  =  120;
        XYBound =  2;
```

Y

YSize

Syntax

YSize = n;

Remarks

Defines the vertical (y) size of a cell matrix

Keep in mind that high XYSize or YSize values are very CPU time-consuming

Example

```
RECIPE  XYSize  =  120;
        YSize =  100;
```

Z

Zet

Syntax

Zet = n;

Remarks

Zet is the number of different states that can be adopted by any cell. The Zet value complies normally with the number of available Colors.

Example

```
RECIPE  XYSize  =  140;
        XYBound = 3 ;
        Zet = 20;
        Colors = 20;
```