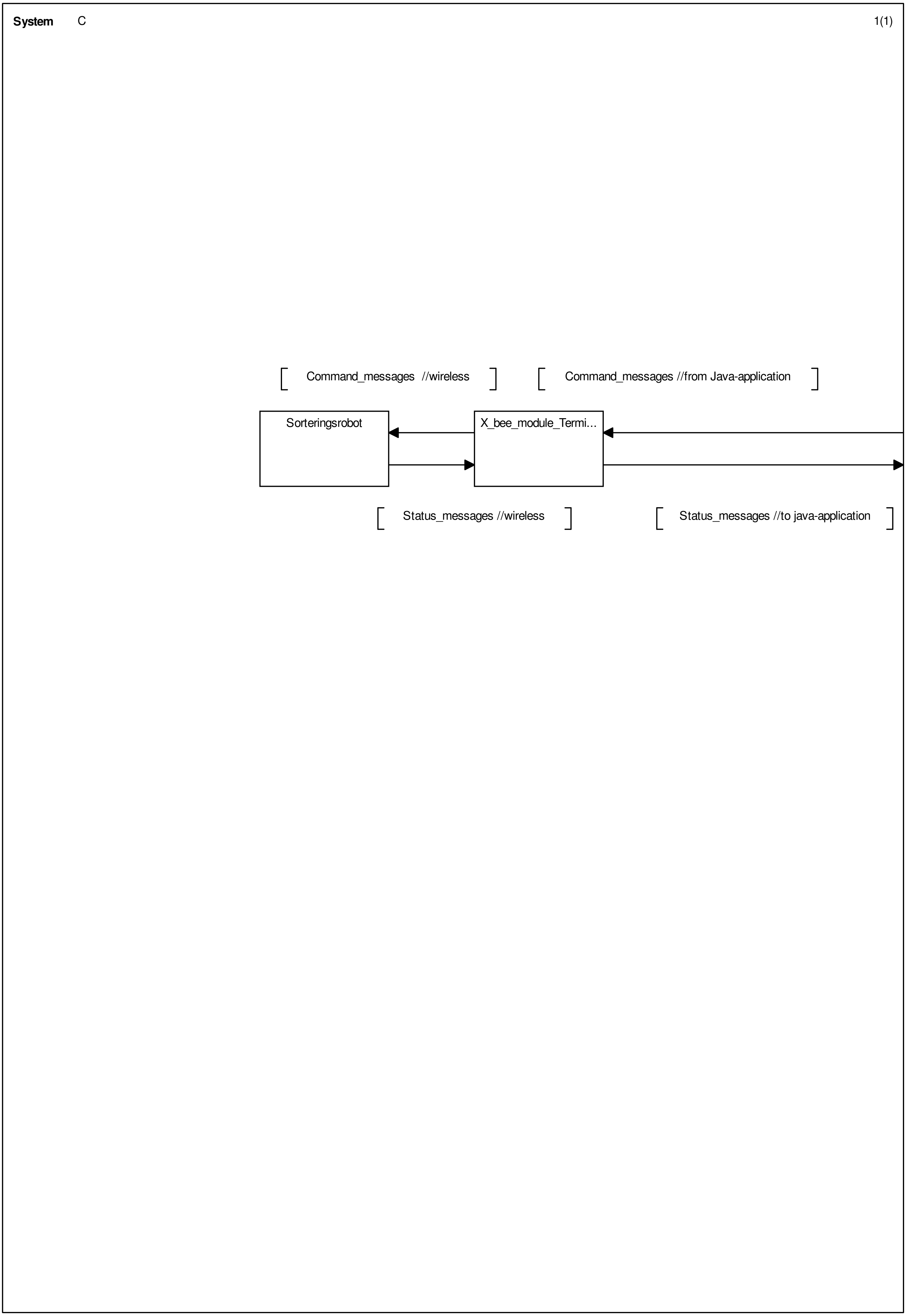
system
predefined


System C
1(1)
[ Command_messages //wireless ]
[ Command_messages //from Java-application ]
Sorteringsrobot
X_bee_module_Termi...
[ Status_messages //wireless ]
[ Status_messages //to java-application ]

**Block** Sorteringsrobot

1(1)

```
newType LocationType
 literals
L_Red,L_Green,L_Blue,L_Other,L_Deli...
endnewtype LocationType;
```

```
newType ClawType
 literals Grip,Release
endnewtype ClawType;
```

```
newType ColorType
 literals
C_Red,C_Green,C_Blue,C_Other,
C_None
endnewtype ColorType;
```

```
newType LineType
 literals
L_Straight,L_Right,L_Left,L_XtrRight,L...
endnewtype  LineType;
```

```
newType JunctionType
 literals J_Right,J_Left,J_Tjunction
endnewtype JunctionType;
```

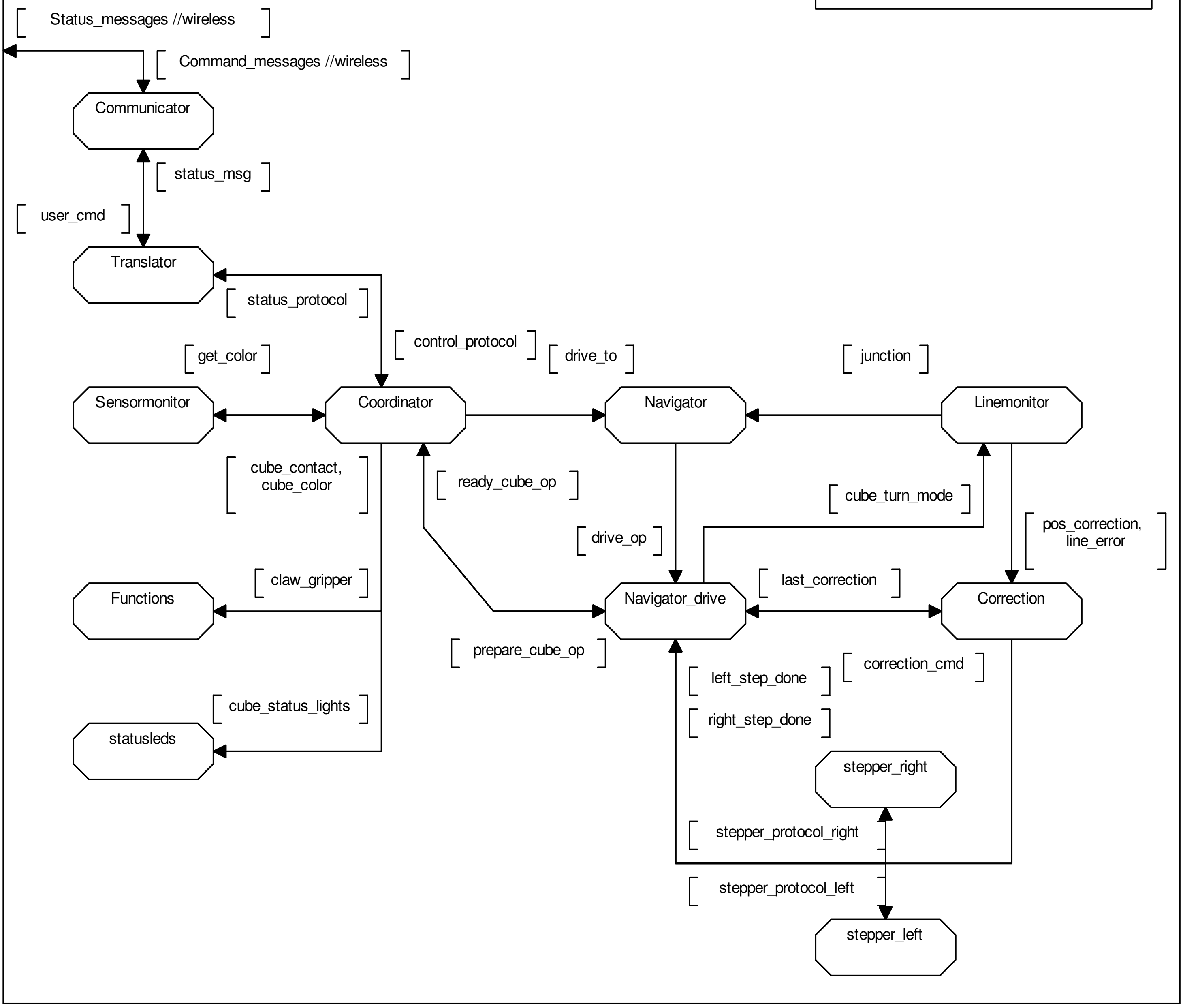
```
newType DriveType
 literals
Right,Left,Turn,Rotate,Forward,Backw...
endnewtype DriveType;
```

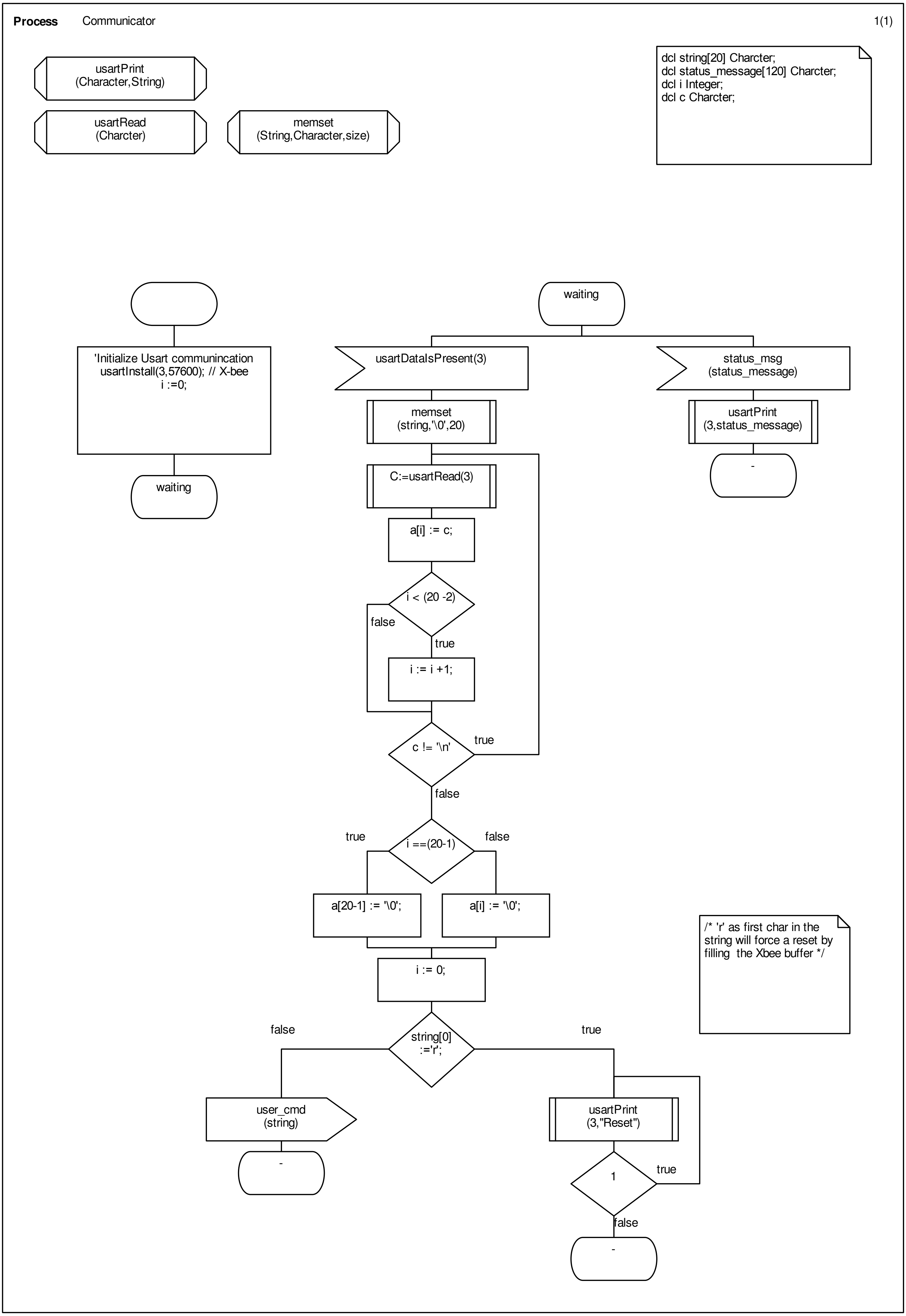
```
newType StepperType
 literals S_Still, S_Forward,
S_Stepforward, S_Stepbackward
endnewtype StepperType;
```

```
newType CorrectioncmdType
 literals stop, start
endnewType CorrectioncmdType;
```

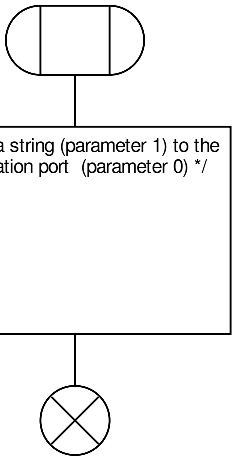
```
newType CubeturnmodeType
 literals on, off
endnewtype CubeturnmodeType;
```

```
SIGNAL
 user_cmd (Charstring),
 status_msg (Charstring),
 status_protocol(Character, Integer, Integer,
Integer, Integer, Integer),
 control_protocol(Character, Integer, Integer,
Integer, Integer, Integer),
 get_color,
 cube_contact,
 cube_color(ColorType),
 claw_gripper(ClawType),
 cube_status_lights(ColorType),
 drive_to(LocationType),
 prepare_cube_op,
 ready_cube_op,
 cube_turn_mode(CubeturnmodeType),
 cube_correction_angle_offset(Float, Float),
 junction(JunctionType),
 pos_correction(LineType),
 last_correction(LineType),
 line_error,
 drive_op(DriveType),
 stepper_protocol_left(StepperType, Character,
Integer),
 stepper_protocol_right(StepperType, Character,
Integer),
 left_step_done,
 right_step_done,
 correction_cmd(CorrectioncmdType);
```



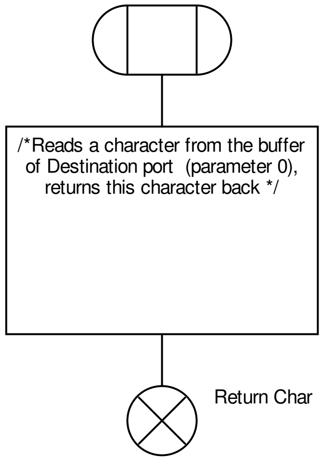

Process Communicator
1(1)
usartPrint (Character,String)
usartRead (Charcter)
memset (String,Character,size)
dcl string[20] Charcter;
dcl status_message[120] Charcter;
dcl i Integer;
dcl c Charcter;
'Initialize Usart communincation
usartInstall(3,57600); // X-bee
i :=0;
waiting
waiting
usartDataIsPresent(3)
memset (string,'\0',20)
C:=usartRead(3)
a[i] := c;
i < (20 -2)
false
true
i := i +1;
c != '\n'
true
false
true
i ==(20-1)
false
a[20-1] := '\0';
a[i] := '\0';
i := 0;
false
string[0] :='r';
true
user_cmd (string)
-
usartPrint (3,"Reset")
1
true
false
-
status_msg (status_message)
usartPrint (3,status_message)
-
/* 'r' as first char in the string will force a reset by filling the Xbee buffer */

**Procedure** usartPrint 1(1)


/*Write a string (parameter 1) to the Destination port (parameter 0) */

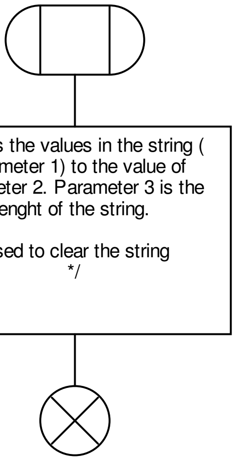
**Procedure** usartRead

1(1)



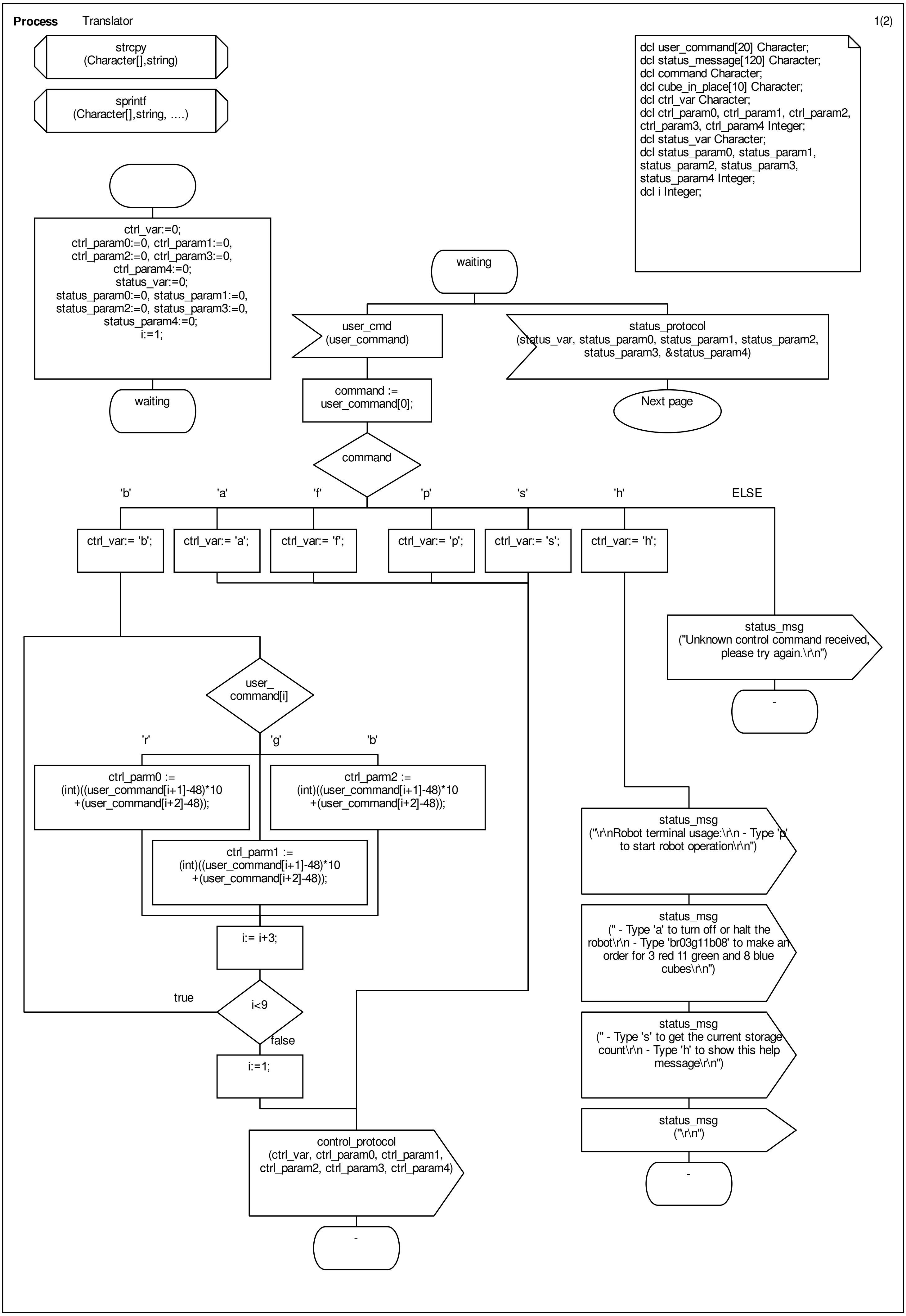
/*Reads a character from the buffer of Destination port (parameter 0), returns this character back */

**Procedure** memset 1(1)

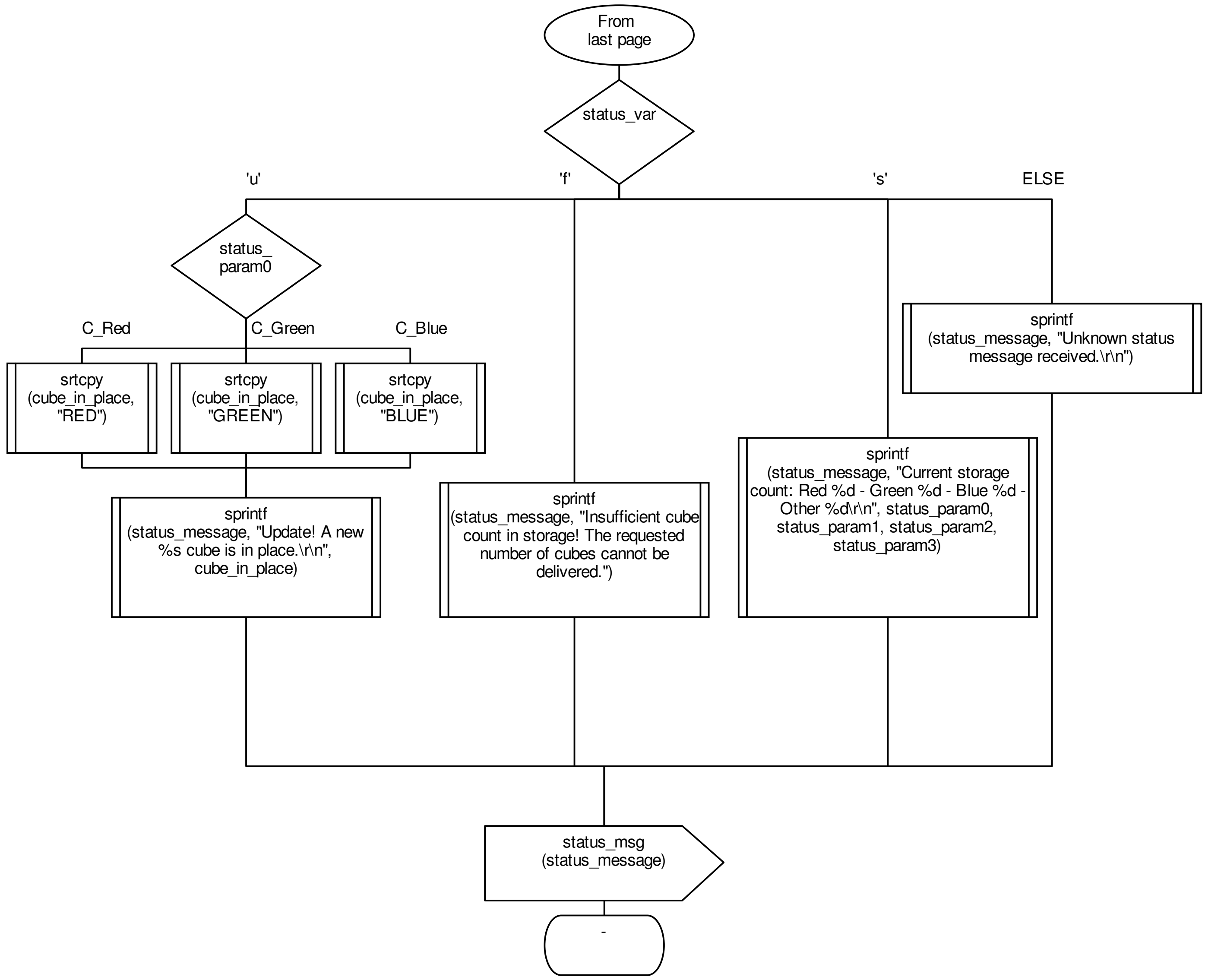


/* Sets the values in the string ( parameter 1) to the value of parameter 2. Parameter 3 is the lenght of the string.

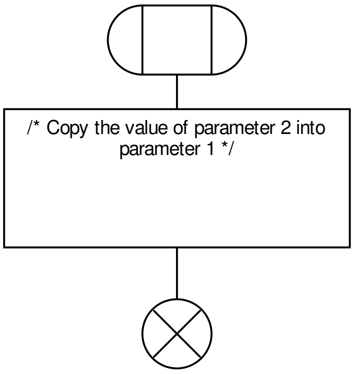
Used to clear the string
*/


Process Translator
1(2)
strcpy
(Character[],string)
sprintf
(Character[],string, ....)
dcl user_command[20] Character;
dcl status_message[120] Character;
dcl command Character;
dcl cube_in_place[10] Character;
dcl ctrl_var Character;
dcl ctrl_param0, ctrl_param1, ctrl_param2, ctrl_param3, ctrl_param4 Integer;
dcl status_var Character;
dcl status_param0, status_param1, status_param2, status_param3, status_param4 Integer;
dcl i Integer;
ctrl_var:=0;
ctrl_param0:=0, ctrl_param1:=0,
ctrl_param2:=0, ctrl_param3:=0,
ctrl_param4:=0;
status_var:=0;
status_param0:=0, status_param1:=0,
status_param2:=0, status_param3:=0,
status_param4:=0;
i:=1;
waiting
waiting
user_cmd
(user_command)
status_protocol
(status_var, status_param0, status_param1, status_param2, status_param3, &status_param4)
Next page
command :=
user_command[0];
command
'b'
'a'
'f'
'p'
's'
'h'
ELSE
ctrl_var:= 'b';
ctrl_var:= 'a';
ctrl_var:= 'f';
ctrl_var:= 'p';
ctrl_var:= 's';
ctrl_var:= 'h';
status_msg
("Unknown control command received, please try again.\r\n")
-
user_
command[i]
'r'
'g'
'b'
ctrl_parm0 :=
(int)((user_command[i+1]-48)*10
+(user_command[i+2]-48));
ctrl_parm2 :=
(int)((user_command[i+1]-48)*10
+(user_command[i+2]-48));
ctrl_parm1 :=
(int)((user_command[i+1]-48)*10
+(user_command[i+2]-48));
i:= i+3;
true
i<9
false
i:=1;
control_protocol
(ctrl_var, ctrl_param0, ctrl_param1, ctrl_param2, ctrl_param3, ctrl_param4)
-
status_msg
("\r\nRobot terminal usage:\r\n - Type 'p' to start robot operation\r\n")
status_msg
(" - Type 'a' to turn off or halt the robot\r\n - Type 'br03g11b08' to make an order for 3 red 11 green and 8 blue cubes\r\n")
status_msg
(" - Type 's' to get the current storage count\r\n - Type 'h' to show this help message\r\n")
status_msg
("\r\n")
-

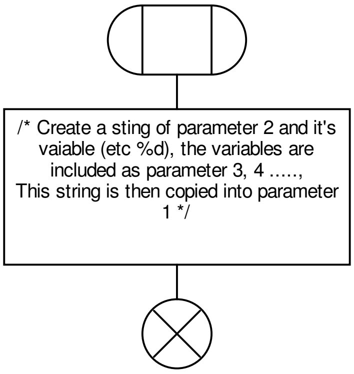
**Process** Translator 2(2)


From last page
status_var
'u'
'f'
's'
ELSE
status_ param0
C_Red
C_Green
C_Blue
srtcpy (cube_in_place, "RED")
srtcpy (cube_in_place, "GREEN")
srtcpy (cube_in_place, "BLUE")
sprintf (status_message, "Update! A new %s cube is in place.\r\n", cube_in_place)
sprintf (status_message, "Insufficient cube count in storage! The requested number of cubes cannot be delivered.")
sprintf (status_message, "Current storage count: Red %d - Green %d - Blue %d - Other %d\r\n", status_param0, status_param1, status_param2, status_param3)
sprintf (status_message, "Unknown status message received.\r\n")
status_msg (status_message)
-

**Procedure** strcpy 1(1)

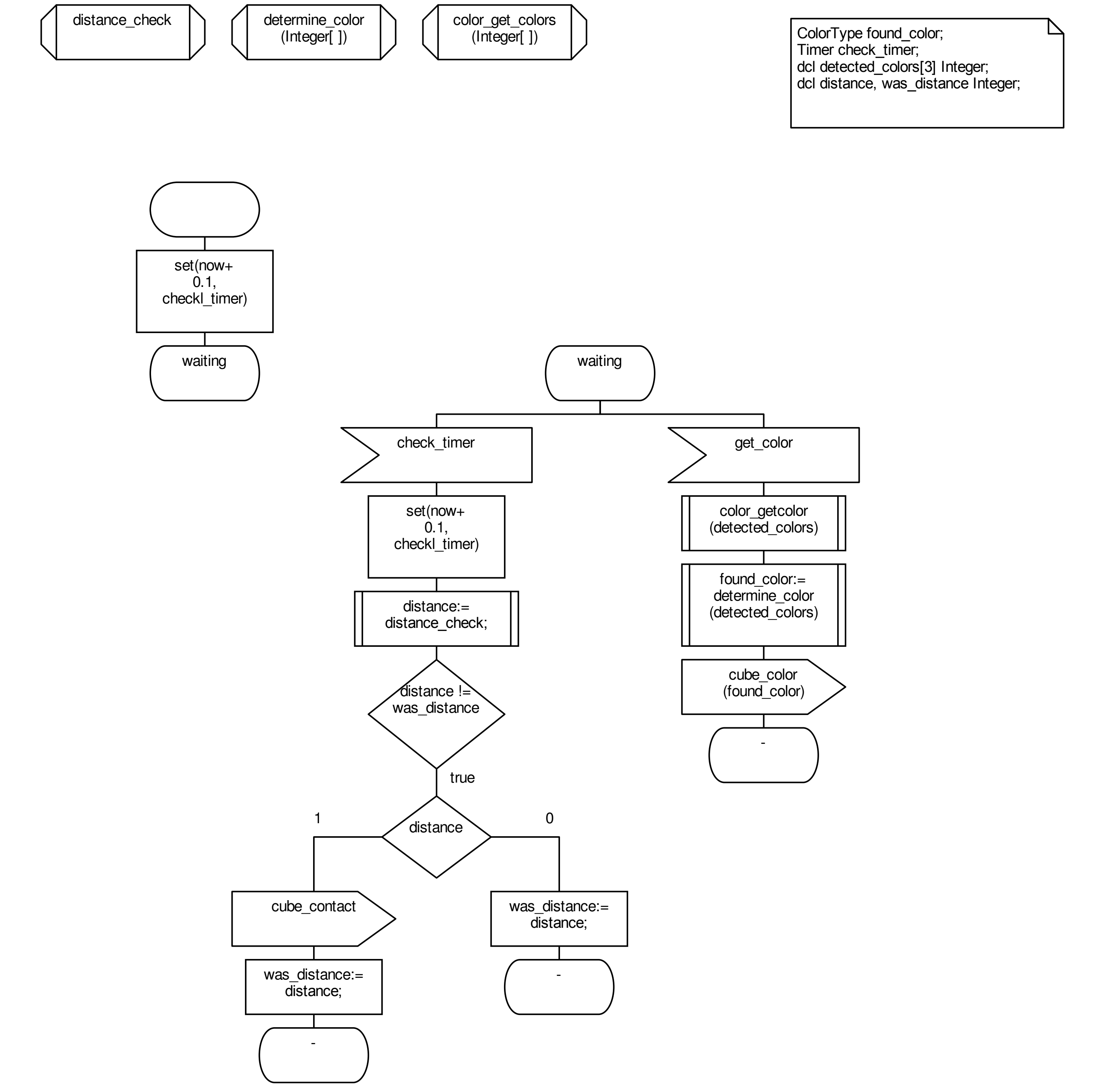


**Procedure** sprintf 1(1)

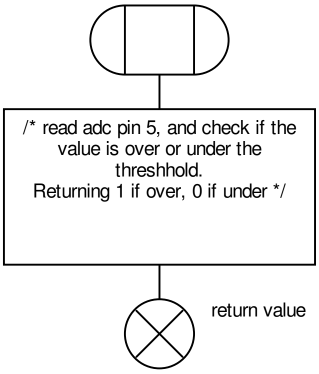


/* Create a sting of parameter 2 and it's
vaiable (etc %d), the variables are
included as parameter 3, 4 .....,
This string is then copied into parameter
1 */

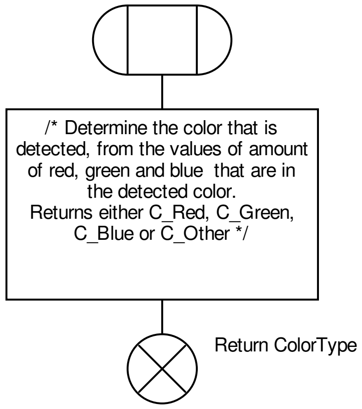
**Process** Sensormonitor 1(1)


distance_check
determine_color (Integer[ ])
color_get_colors (Integer[ ])
ColorType found_color;
Timer check_timer;
dcl detected_colors[3] Integer;
dcl distance, was_distance Integer;
set(now+ 0.1, checkl_timer)
waiting
waiting
check_timer
get_color
set(now+ 0.1, checkl_timer)
distance:= distance_check;
distance != was_distance
true
distance
1
0
cube_contact
was_distance:= distance;
-
was_distance:= distance;
-
color_getcolor (detected_colors)
found_color:= determine_color (detected_colors)
cube_color (found_color)
-

**Procedure** distance_check 1(1)

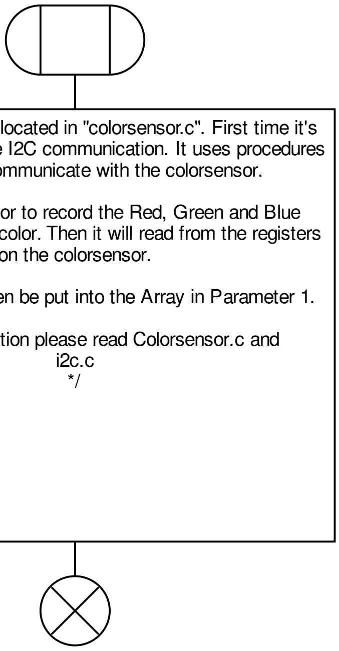

/* read adc pin 5, and check if the value is over or under the threshhold.
Returning 1 if over, 0 if under */
return value

**Procedure** determine_color 1(1)



/* Determine the color that is detected, from the values of amount of red, green and blue that are in the detected color.
Returns either C_Red, C_Green, C_Blue or C_Other */

**Procedure** color_get_colors 1(1)



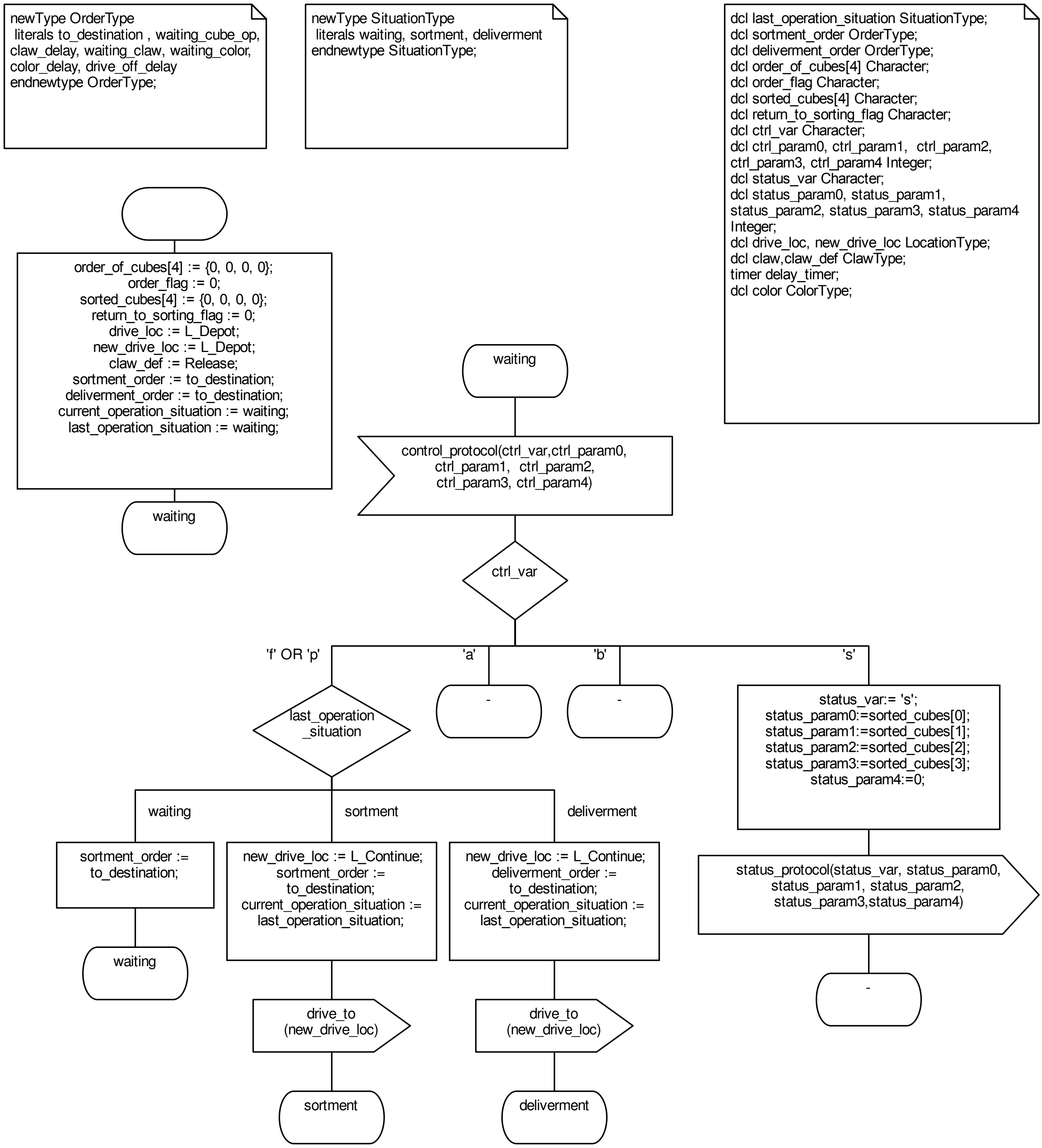
/* This Procedure is located in "colorsensor.c". First time it's called it will set up the I2C communication. It uses procedures in "i2c.c" to communicate with the colorsensor.

It will ask the sensor to record the Red, Green and Blue values of the current color. Then it will read from the registers on the colorsensor.

These values will then be put into the Array in Parameter 1.

For better inspection please read Colorsensor.c and i2c.c
*/

**Process** Coordinator 1(8)



```
newType OrderType
 literals to_destination , waiting_cube_op,
claw_delay, waiting_claw, waiting_color,
color_delay, drive_off_delay
endnewtype OrderType;
```

```
newType SituationType
 literals waiting, sortment, deliverment
endnewtype SituationType;
```

```
dcl last_operation_situation SituationType;
dcl sortment_order OrderType;
dcl deliverment_order OrderType;
dcl order_of_cubes[4] Character;
dcl order_flag Character;
dcl sorted_cubes[4] Character;
dcl return_to_sorting_flag Character;
dcl ctrl_var Character;
dcl ctrl_param0, ctrl_param1,  ctrl_param2,
ctrl_param3, ctrl_param4 Integer;
dcl status_var Character;
dcl status_param0, status_param1,
status_param2, status_param3, status_param4
Integer;
dcl drive_loc, new_drive_loc LocationType;
dcl claw,claw_def ClawType;
timer delay_timer;
dcl color ColorType;
```

```
order_of_cubes[4] := {0, 0, 0, 0};
order_flag := 0;
sorted_cubes[4] := {0, 0, 0, 0};
return_to_sorting_flag := 0;
drive_loc := L_Depot;
new_drive_loc := L_Depot;
claw_def := Release;
sortment_order := to_destination;
deliverment_order := to_destination;
current_operation_situation := waiting;
last_operation_situation := waiting;
```

```
control_protocol(ctrl_var,ctrl_param0,
ctrl_param1,  ctrl_param2,
ctrl_param3, ctrl_param4)
```

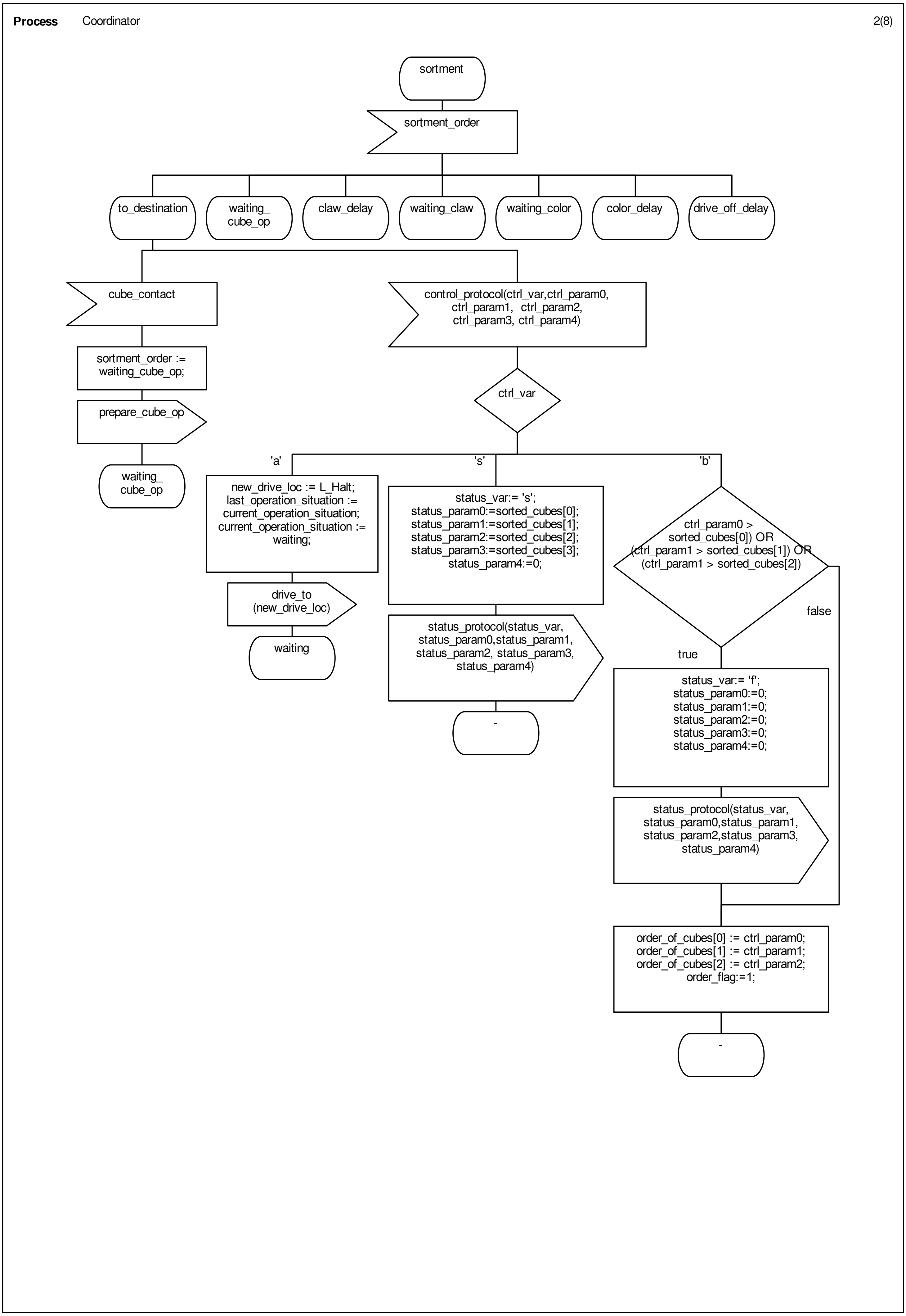
```
status_var:= 's';
status_param0:=sorted_cubes[0];
status_param1:=sorted_cubes[1];
status_param2:=sorted_cubes[2];
status_param3:=sorted_cubes[3];
status_param4:=0;
```

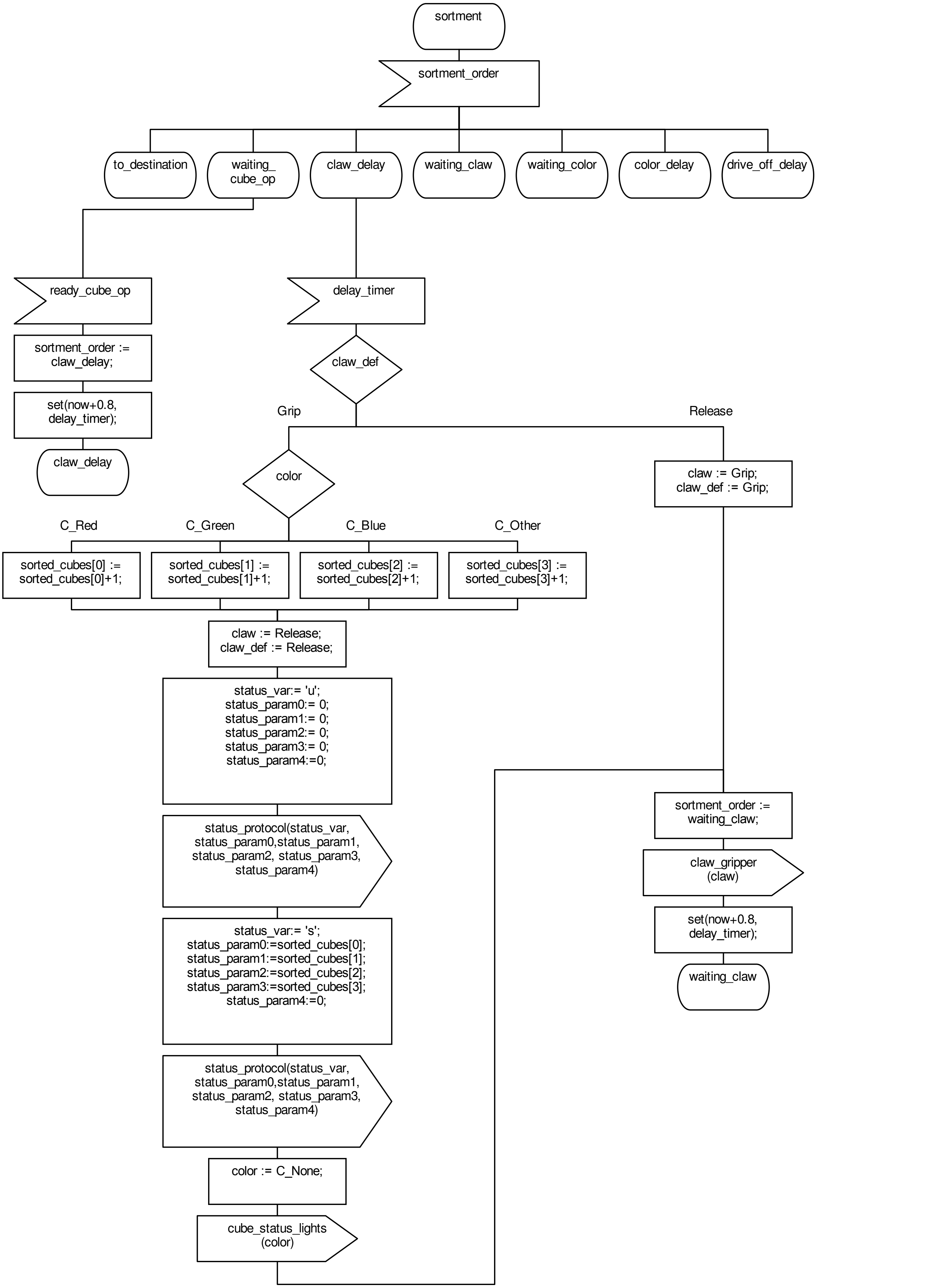
```
status_protocol(status_var, status_param0,
status_param1, status_param2,
status_param3,status_param4)
```

```
sortment_order :=
to_destination;
```

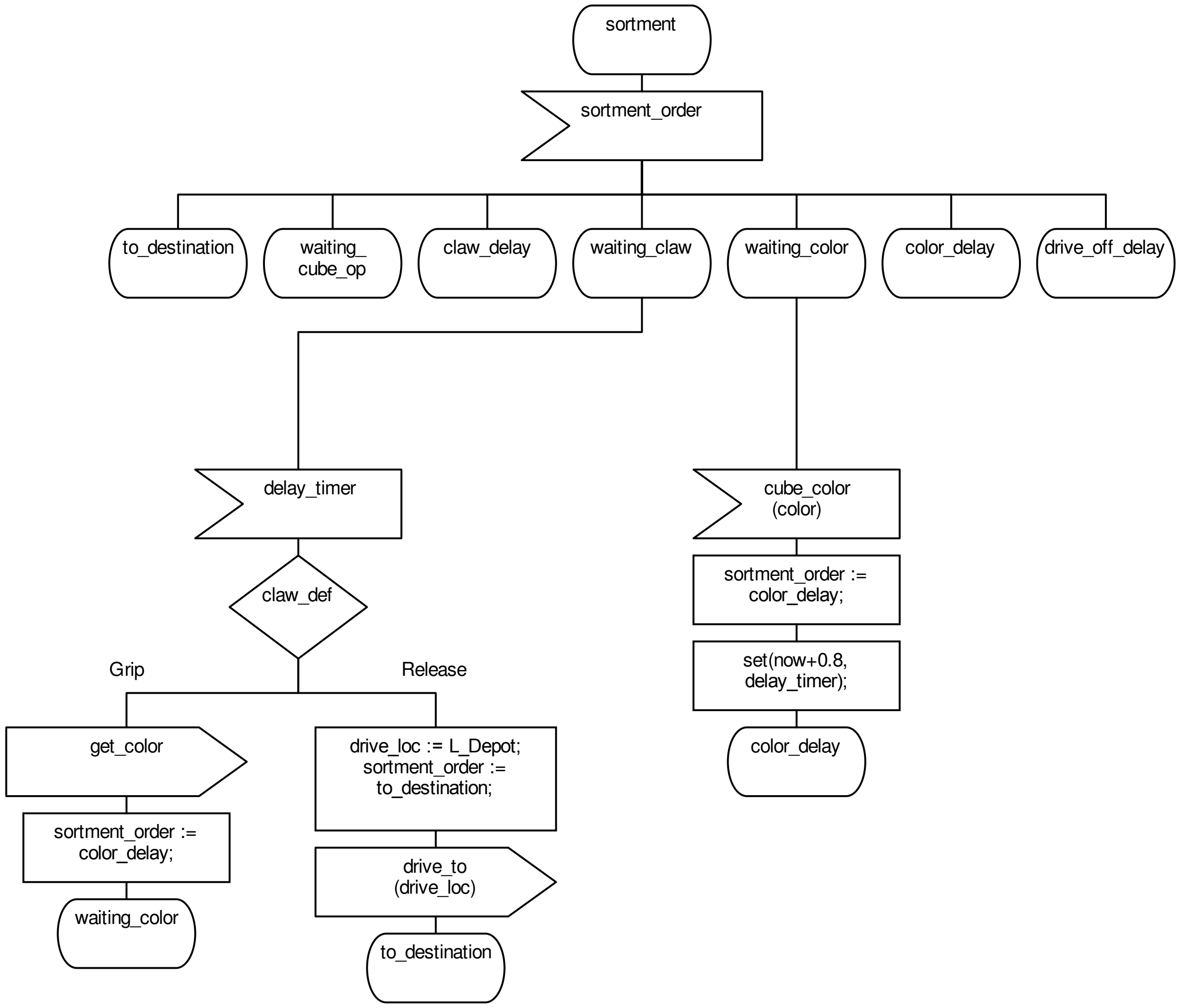
```
new_drive_loc := L_Continue;
sortment_order :=
to_destination;
current_operation_situation :=
last_operation_situation;
```

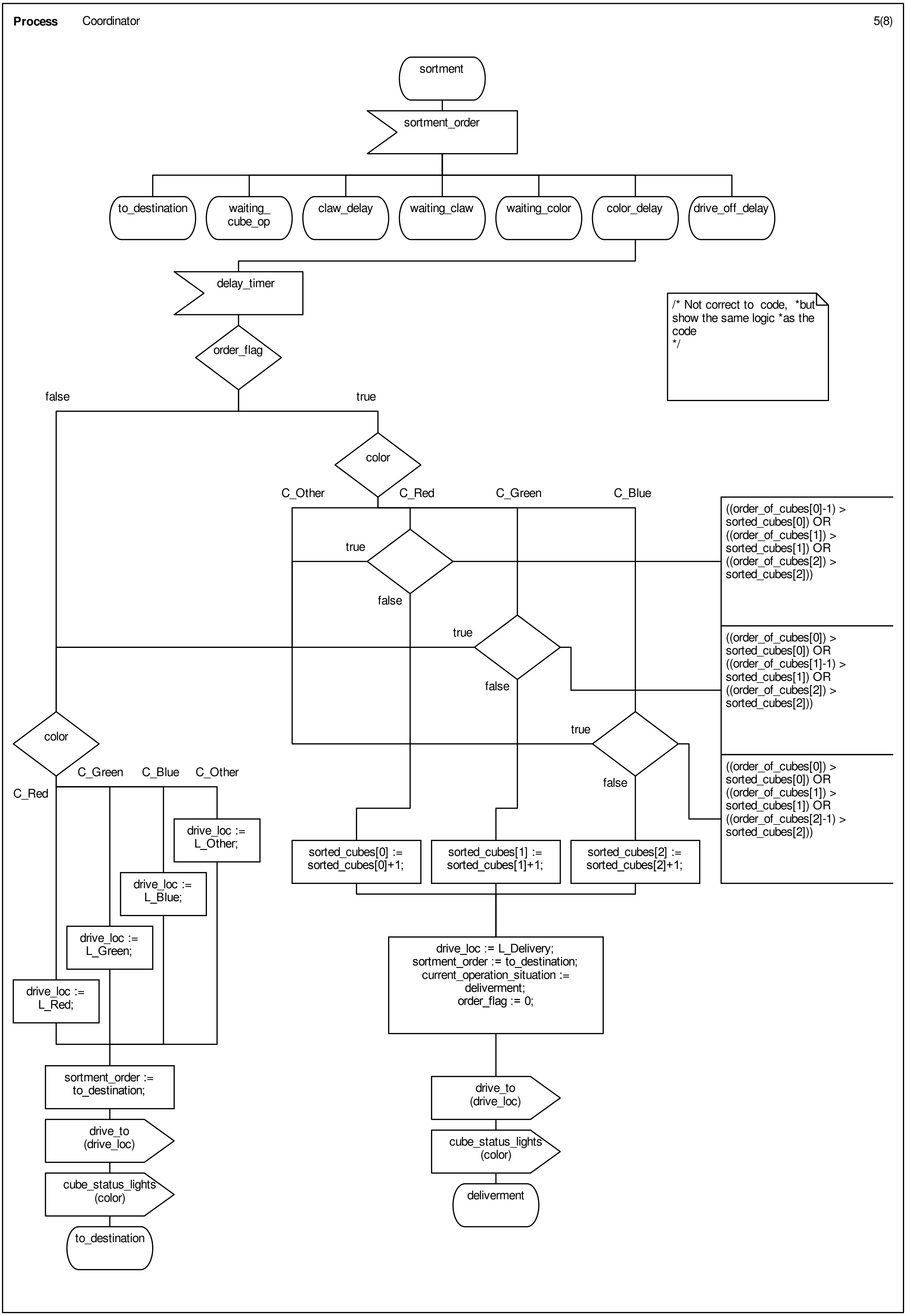
```
new_drive_loc := L_Continue;
deliverment_order :=
to_destination;
current_operation_situation :=
last_operation_situation;
```


Process
Coordinator
2(8)
sortment
sortment_order
to_destination
waiting_
cube_op
claw_delay
waiting_claw
waiting_color
color_delay
drive_off_delay
cube_contact
sortment_order :=
waiting_cube_op;
prepare_cube_op
waiting_
cube_op
control_protocol(ctrl_var,ctrl_param0,
ctrl_param1, ctrl_param2,
ctrl_param3, ctrl_param4)
ctrl_var
'a'
's'
'b'
new_drive_loc := L_Halt;
last_operation_situation :=
current_operation_situation;
current_operation_situation :=
waiting;
drive_to
(new_drive_loc)
waiting
status_var:= 's';
status_param0:=sorted_cubes[0];
status_param1:=sorted_cubes[1];
status_param2:=sorted_cubes[2];
status_param3:=sorted_cubes[3];
status_param4:=0;
status_protocol(status_var,
status_param0,status_param1,
status_param2, status_param3,
status_param4)
-
ctrl_param0 >
sorted_cubes[0]) OR
(ctrl_param1 > sorted_cubes[1]) OR
(ctrl_param1 > sorted_cubes[2])
false
true
status_var:= 'f';
status_param0:=0;
status_param1:=0;
status_param2:=0;
status_param3:=0;
status_param4:=0;
status_protocol(status_var,
status_param0,status_param1,
status_param2,status_param3,
status_param4)
order_of_cubes[0] := ctrl_param0;
order_of_cubes[1] := ctrl_param1;
order_of_cubes[2] := ctrl_param2;
order_flag:=1;
-

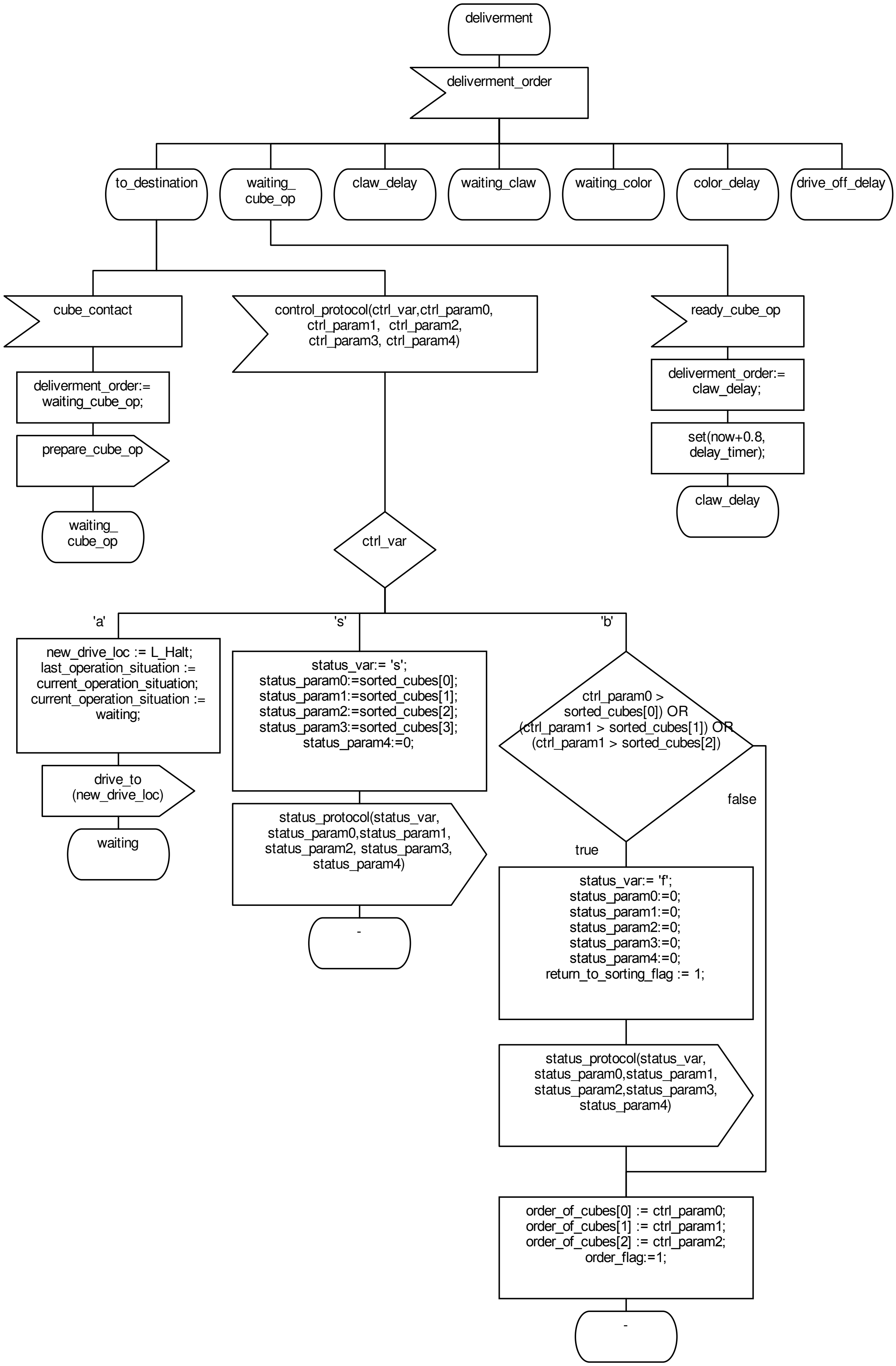

sortment
sortment_order
to_destination
waiting_ cube_op
claw_delay
waiting_claw
waiting_color
color_delay
drive_off_delay
ready_cube_op
sortment_order := claw_delay;
set(now+0.8, delay_timer);
claw_delay
delay_timer
claw_def
Grip
Release
color
C_Red
C_Green
C_Blue
C_Other
sorted_cubes[0] := sorted_cubes[0]+1;
sorted_cubes[1] := sorted_cubes[1]+1;
sorted_cubes[2] := sorted_cubes[2]+1;
sorted_cubes[3] := sorted_cubes[3]+1;
claw := Release; claw_def := Release;
status_var:= 'u'; status_param0:= 0; status_param1:= 0; status_param2:= 0; status_param3:= 0; status_param4:=0;
status_protocol(status_var, status_param0,status_param1, status_param2, status_param3, status_param4)
status_var:= 's'; status_param0:=sorted_cubes[0]; status_param1:=sorted_cubes[1]; status_param2:=sorted_cubes[2]; status_param3:=sorted_cubes[3]; status_param4:=0;
status_protocol(status_var, status_param0,status_param1, status_param2, status_param3, status_param4)
color := C_None;
cube_status_lights (color)
claw := Grip; claw_def := Grip;
sortment_order := waiting_claw;
claw_gripper (claw)
set(now+0.8, delay_timer);
waiting_claw

**Process** Coordinator 4(8)

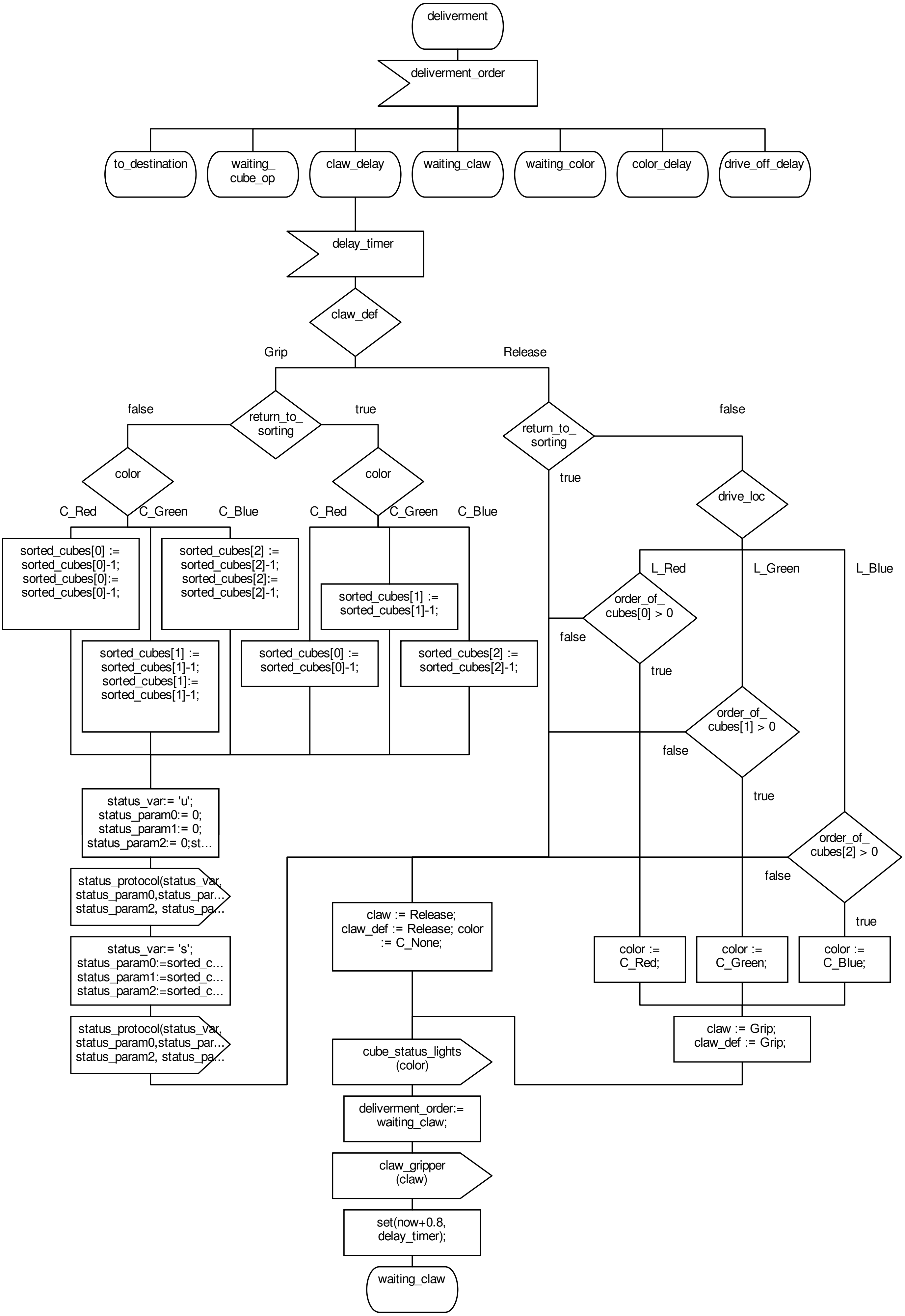


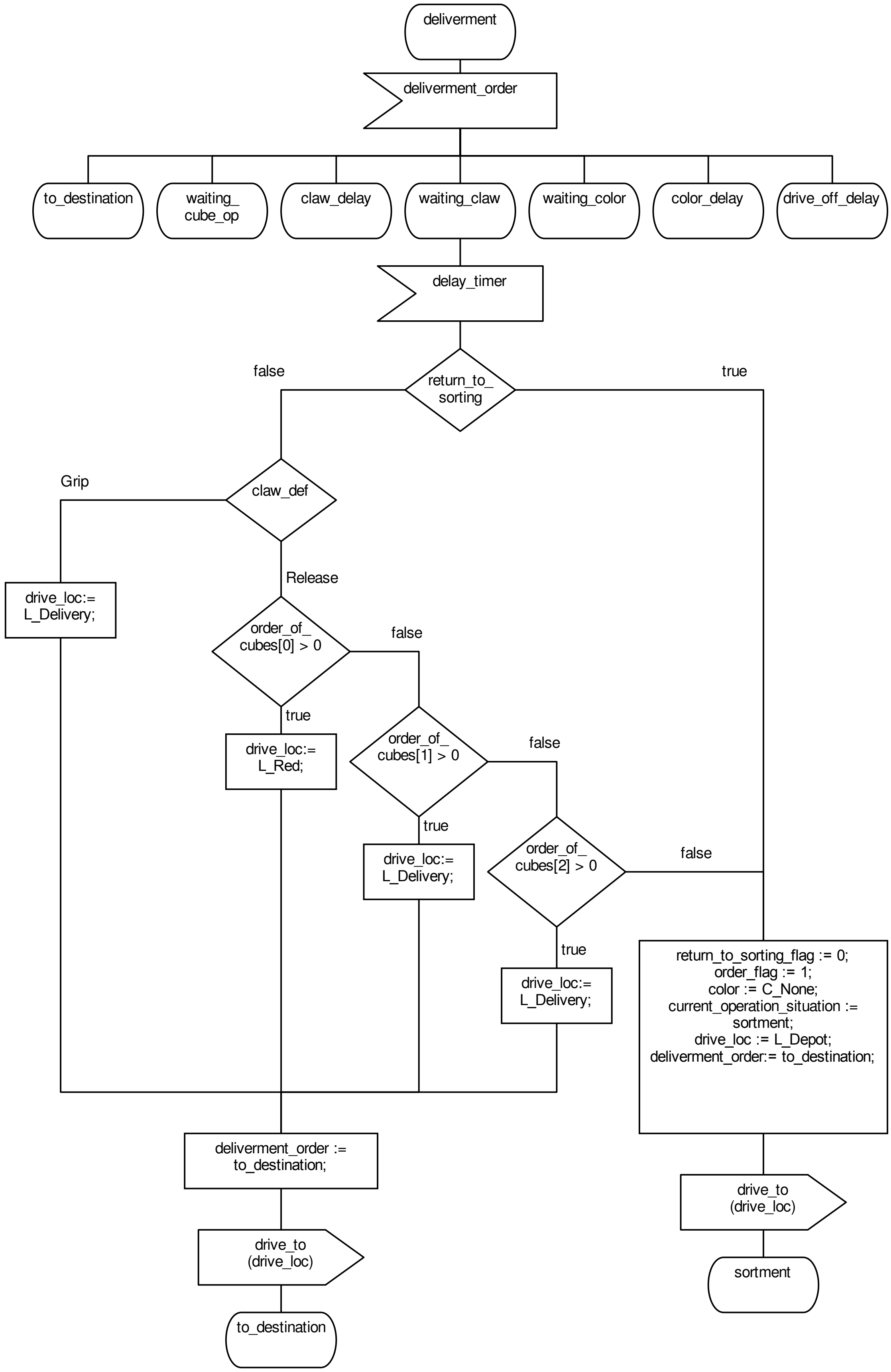

Process
Coordinator
5(8)
sortment
sortment_order
to_destination
waiting_cube_op
claw_delay
waiting_claw
waiting_color
color_delay
drive_off_delay
delay_timer
/* Not correct to code, *but show the same logic *as the code */
order_flag
false
true
color
C_Other
C_Red
C_Green
C_Blue
((order_of_cubes[0]-1) > sorted_cubes[0]) OR ((order_of_cubes[1]) > sorted_cubes[1]) OR ((order_of_cubes[2]) > sorted_cubes[2]))
true
false
true
false
((order_of_cubes[0]) > sorted_cubes[0]) OR ((order_of_cubes[1]-1) > sorted_cubes[1]) OR ((order_of_cubes[2]) > sorted_cubes[2]))
true
false
((order_of_cubes[0]) > sorted_cubes[0]) OR ((order_of_cubes[1]) > sorted_cubes[1]) OR ((order_of_cubes[2]-1) > sorted_cubes[2]))
color
C_Red
C_Green
C_Blue
C_Other
drive_loc := L_Other;
drive_loc := L_Blue;
drive_loc := L_Green;
drive_loc := L_Red;
sorted_cubes[0] := sorted_cubes[0]+1;
sorted_cubes[1] := sorted_cubes[1]+1;
sorted_cubes[2] := sorted_cubes[2]+1;
drive_loc := L_Delivery; sortment_order := to_destination; current_operation_situation := deliverment; order_flag := 0;
sortment_order := to_destination;
drive_to (drive_loc)
cube_status_lights (color)
to_destination
drive_to (drive_loc)
cube_status_lights (color)
deliverment

**Process** Coordinator 6(8)


deliverment
deliverment_order
to_destination
waiting_ cube_op
claw_delay
waiting_claw
waiting_color
color_delay
drive_off_delay
cube_contact
deliverment_order:= waiting_cube_op;
prepare_cube_op
waiting_ cube_op
control_protocol(ctrl_var,ctrl_param0, ctrl_param1, ctrl_param2, ctrl_param3, ctrl_param4)
ctrl_var
ready_cube_op
deliverment_order:= claw_delay;
set(now+0.8, delay_timer);
claw_delay
'a'
's'
'b'
new_drive_loc := L_Halt; last_operation_situation := current_operation_situation; current_operation_situation := waiting;
drive_to (new_drive_loc)
waiting
status_var:= 's'; status_param0:=sorted_cubes[0]; status_param1:=sorted_cubes[1]; status_param2:=sorted_cubes[2]; status_param3:=sorted_cubes[3]; status_param4:=0;
status_protocol(status_var, status_param0,status_param1, status_param2, status_param3, status_param4)
-
ctrl_param0 > sorted_cubes[0]) OR (ctrl_param1 > sorted_cubes[1]) OR (ctrl_param1 > sorted_cubes[2])
false
true
status_var:= 'f'; status_param0:=0; status_param1:=0; status_param2:=0; status_param3:=0; status_param4:=0; return_to_sorting_flag := 1;
status_protocol(status_var, status_param0,status_param1, status_param2,status_param3, status_param4)
order_of_cubes[0] := ctrl_param0; order_of_cubes[1] := ctrl_param1; order_of_cubes[2] := ctrl_param2; order_flag:=1;
-

**Process** Coordinator 7(8)


deliverment
deliverment_order
to_destination
waiting_
cube_op
claw_delay
waiting_claw
waiting_color
color_delay
drive_off_delay
delay_timer
claw_def
Grip
Release
false
return_to_
sorting
true
color
C_Red
C_Green
C_Blue
sorted_cubes[0] := sorted_cubes[0]-1; sorted_cubes[0]:= sorted_cubes[0]-1;
sorted_cubes[2] := sorted_cubes[2]-1; sorted_cubes[2]:= sorted_cubes[2]-1;
sorted_cubes[1] := sorted_cubes[1]-1; sorted_cubes[1]:= sorted_cubes[1]-1;
true
color
C_Red
C_Green
C_Blue
sorted_cubes[1] := sorted_cubes[1]-1;
sorted_cubes[0] := sorted_cubes[0]-1;
sorted_cubes[2] := sorted_cubes[2]-1;
return_to_
sorting
false
true
drive_loc
L_Red
L_Green
L_Blue
order_of_
cubes[0] > 0
false
true
order_of_
cubes[1] > 0
false
true
order_of_
cubes[2] > 0
false
true
status_var:= 'u'; status_param0:= 0; status_param1:= 0; status_param2:= 0;st...
status_protocol(status_var, status_param0,status_par... status_param2, status_pa...
status_var:= 's'; status_param0:=sorted_c... status_param1:=sorted_c... status_param2:=sorted_c...
status_protocol(status_var, status_param0,status_par... status_param2, status_pa...
claw := Release; claw_def := Release; color := C_None;
color := C_Red;
color := C_Green;
color := C_Blue;
claw := Grip; claw_def := Grip;
cube_status_lights (color)
deliverment_order:= waiting_claw;
claw_gripper (claw)
set(now+0.8, delay_timer);
waiting_claw

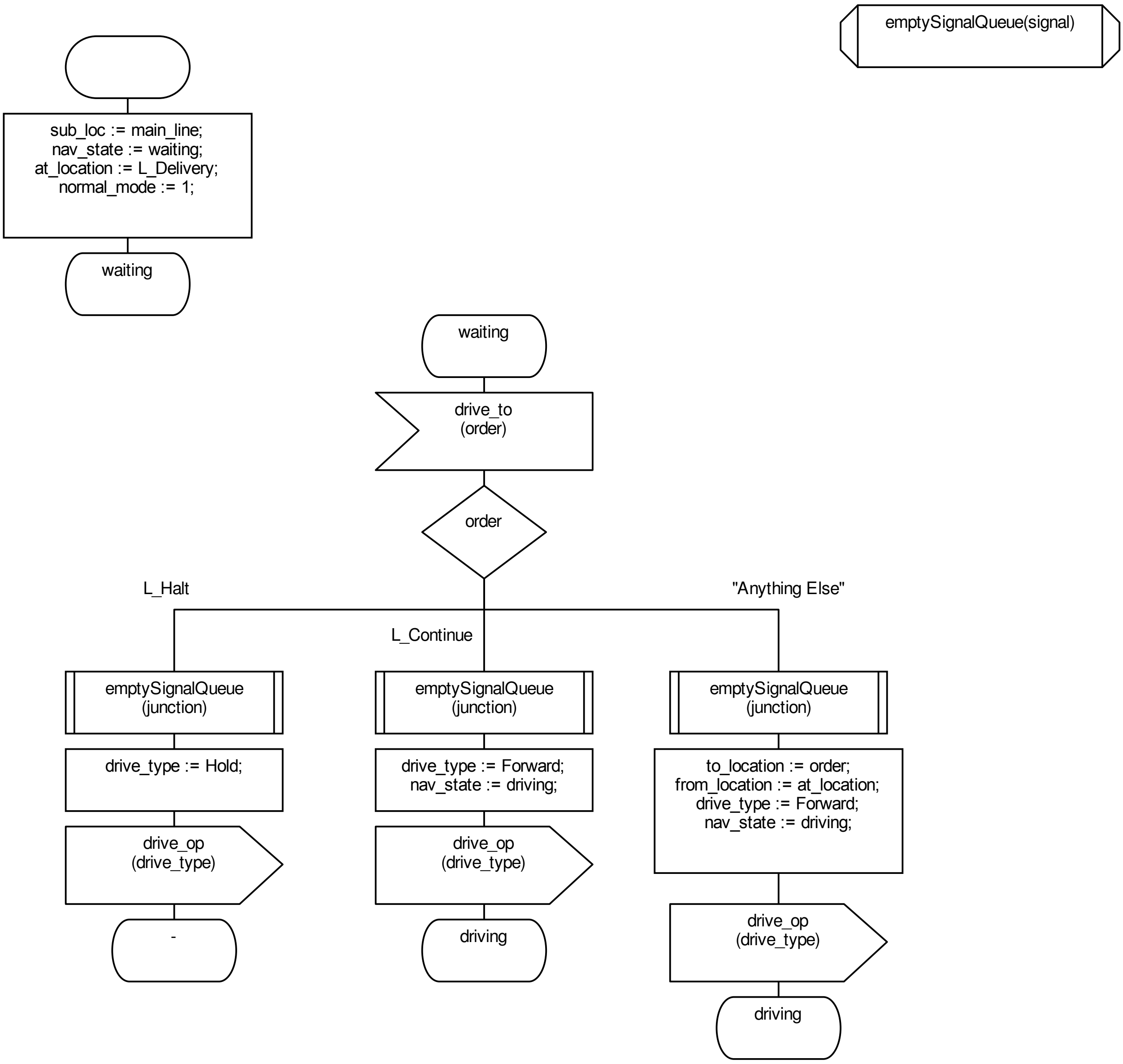

deliverment
deliverment_order
to_destination
waiting_cube_op
claw_delay
waiting_claw
waiting_color
color_delay
drive_off_delay
delay_timer
false
return_to_sorting
true
Grip
claw_def
Release
drive_loc:= L_Delivery;
order_of_cubes[0] > 0
false
true
drive_loc:= L_Red;
order_of_cubes[1] > 0
false
true
drive_loc:= L_Delivery;
order_of_cubes[2] > 0
false
true
drive_loc:= L_Delivery;
return_to_sorting_flag := 0;
order_flag := 1;
color := C_None;
current_operation_situation := sortment;
drive_loc := L_Depot;
deliverment_order:= to_destination;
deliverment_order := to_destination;
drive_to (drive_loc)
to_destination
drive_to (drive_loc)
sortment

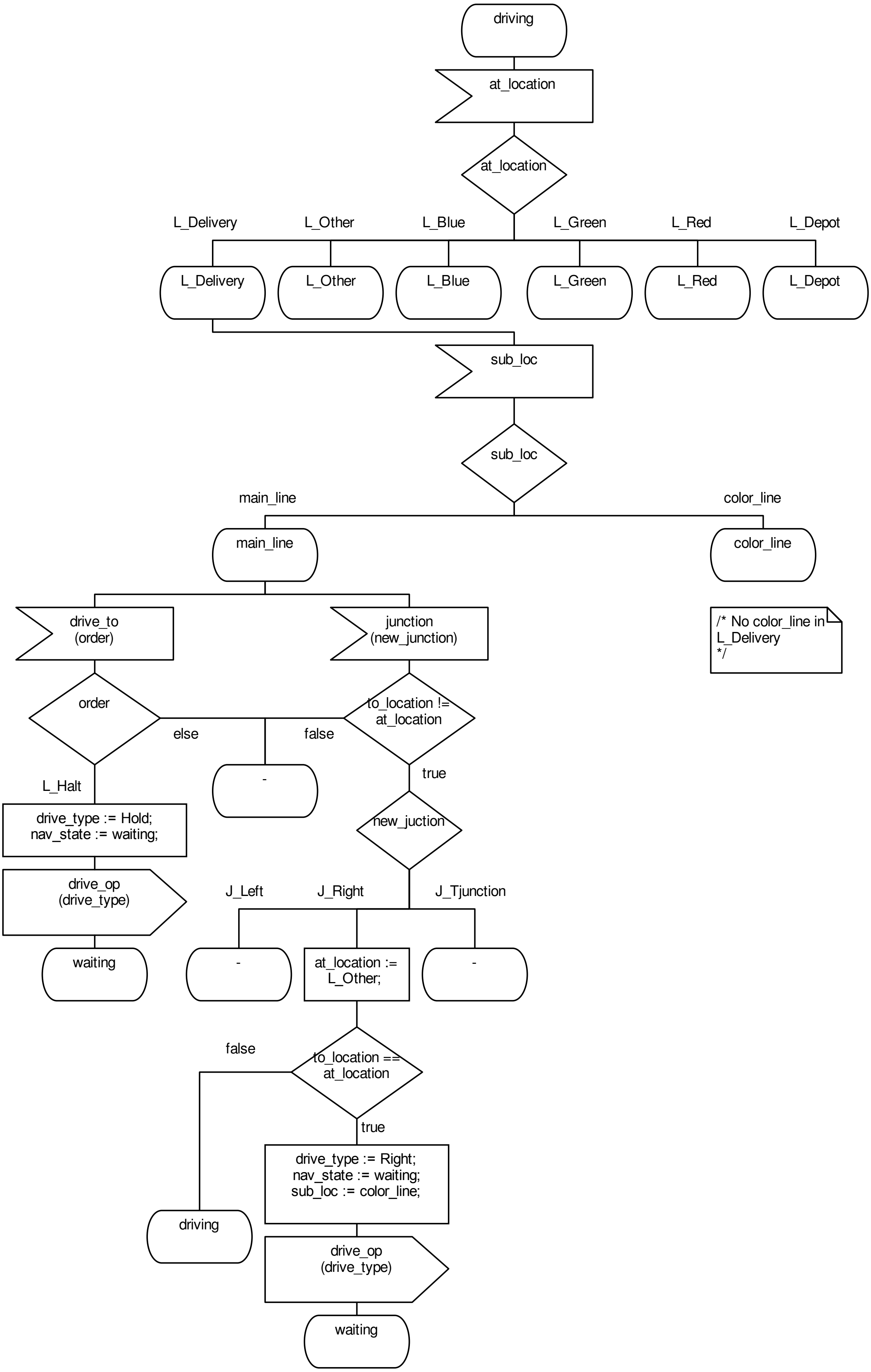
**Process** Navigator 1(7)

```
newType SublocType
 literals main_line, color_line
endnewtype SublocType;
```

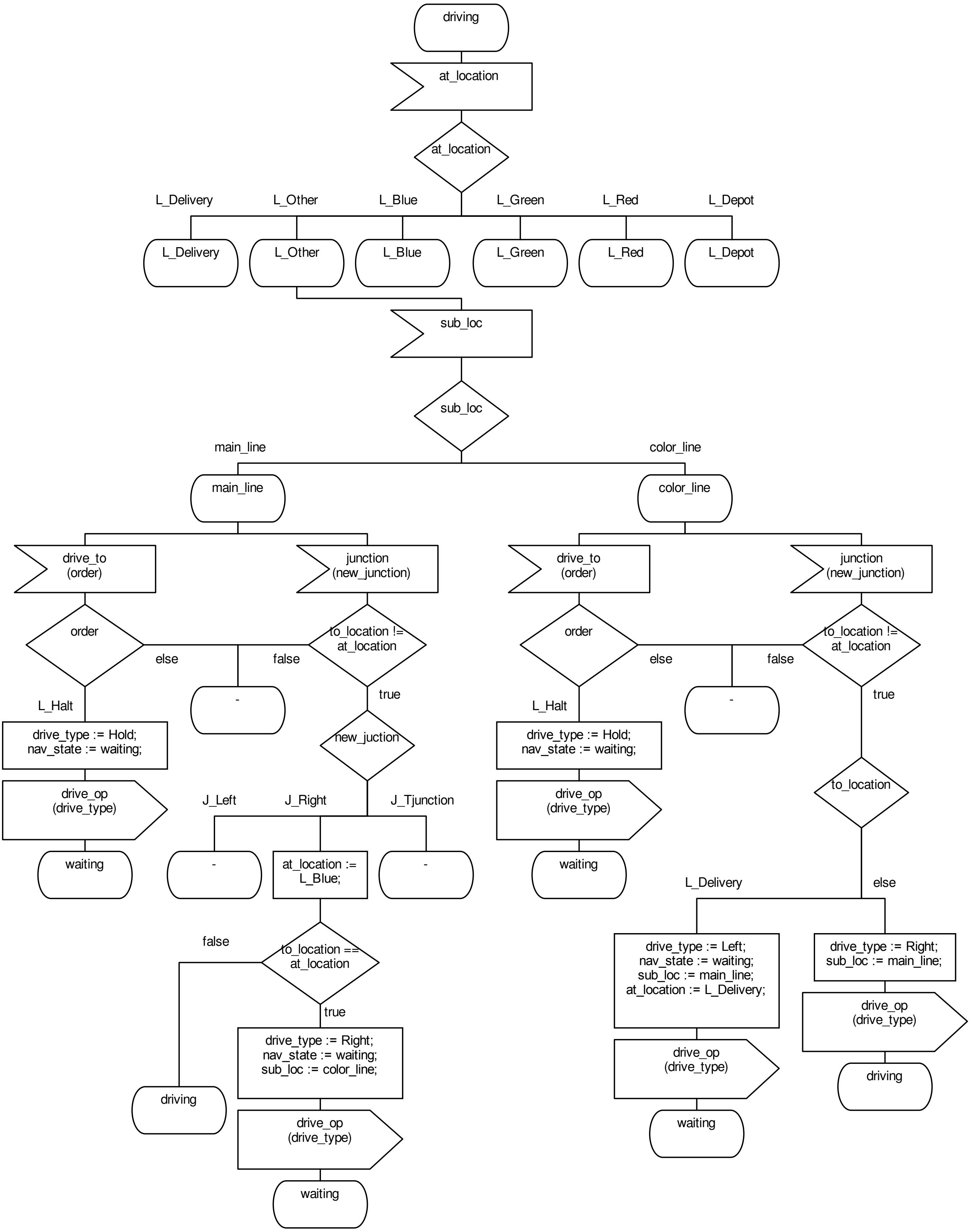
```
newType NavstateType
 literals waiting, driving
endnewtype NavstateType;
```

```
NavstateType navstate;
SublocType sub_loc;
JunctionType new_junction;
LocationType order, to_location,
from_location, at_location;
DriveType drive_type;
dcl Normal_mode Character;
Timer wait_during_turn;
```

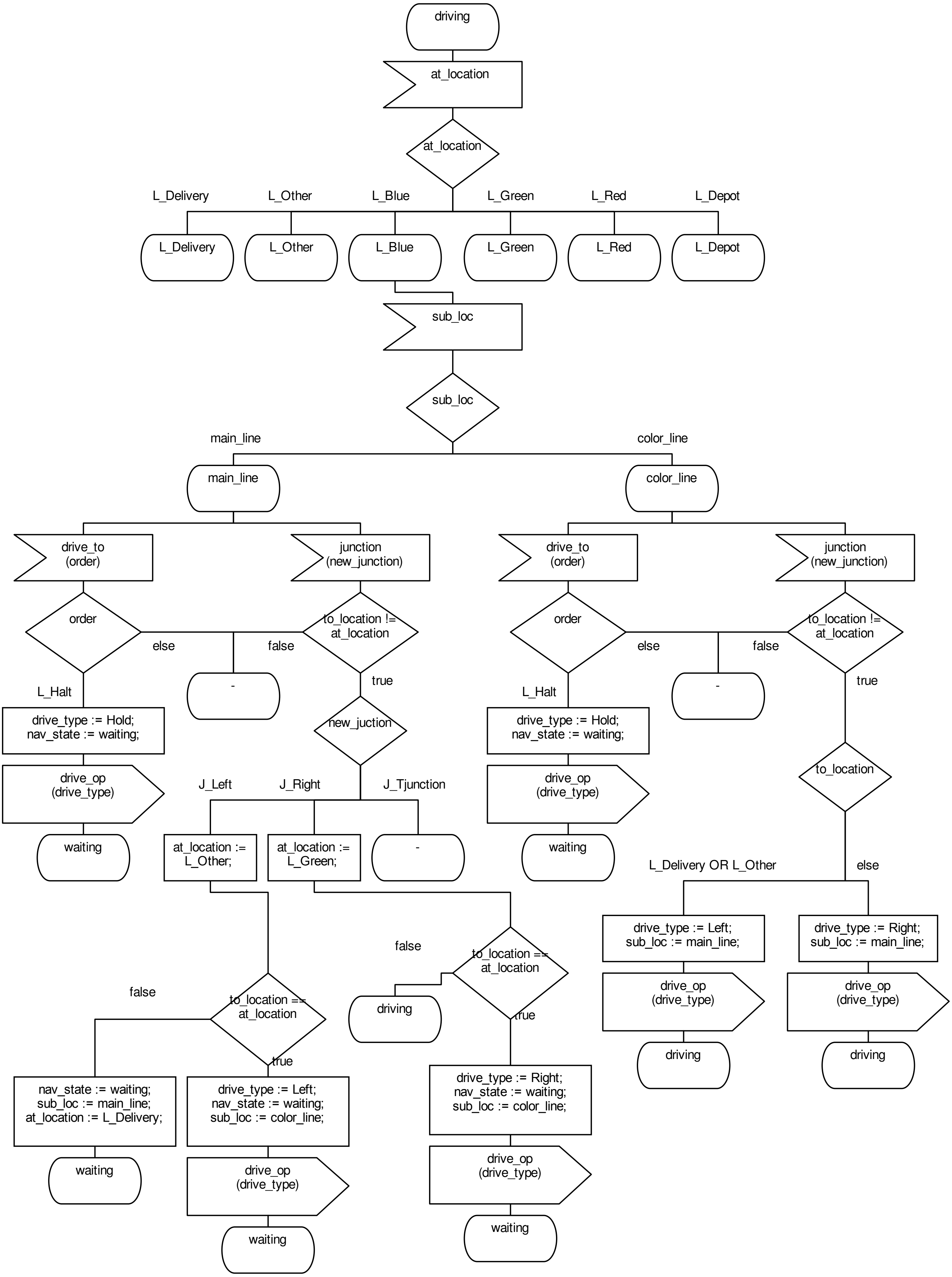


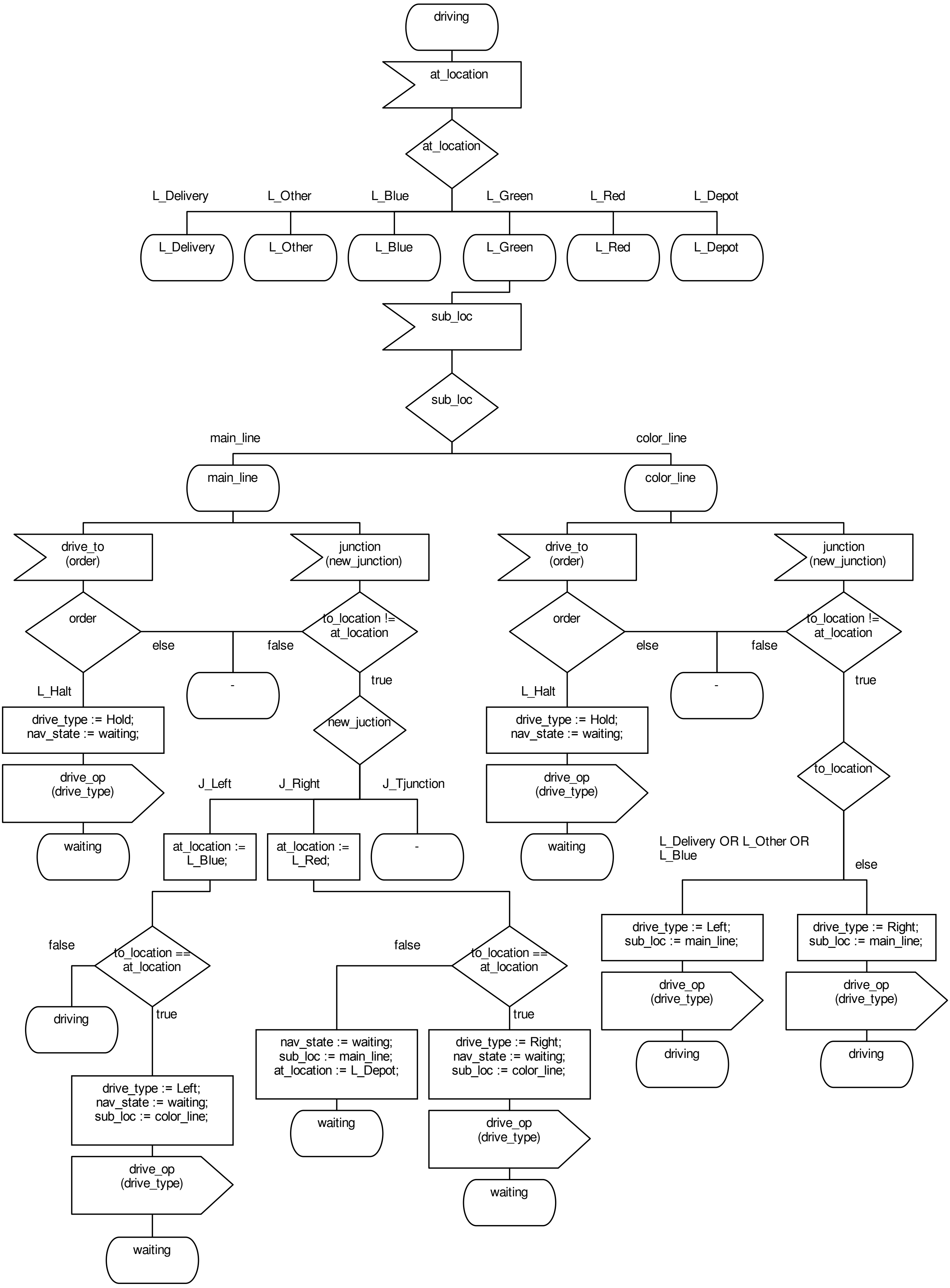

driving
at_location
at_location
L_Delivery
L_Other
L_Blue
L_Green
L_Red
L_Depot
L_Delivery
L_Other
L_Blue
L_Green
L_Red
L_Depot
sub_loc
sub_loc
main_line
color_line
main_line
color_line
drive_to (order)
junction (new_junction)
/* No color_line in L_Delivery */
order
else
false
to_location != at_location
-
true
L_Halt
drive_type := Hold; nav_state := waiting;
new_juction
drive_op (drive_type)
J_Left
J_Right
J_Tjunction
waiting
-
at_location := L_Other;
-
false
to_location == at_location
true
drive_type := Right; nav_state := waiting; sub_loc := color_line;
driving
drive_op (drive_type)
waiting

**Process** Navigator

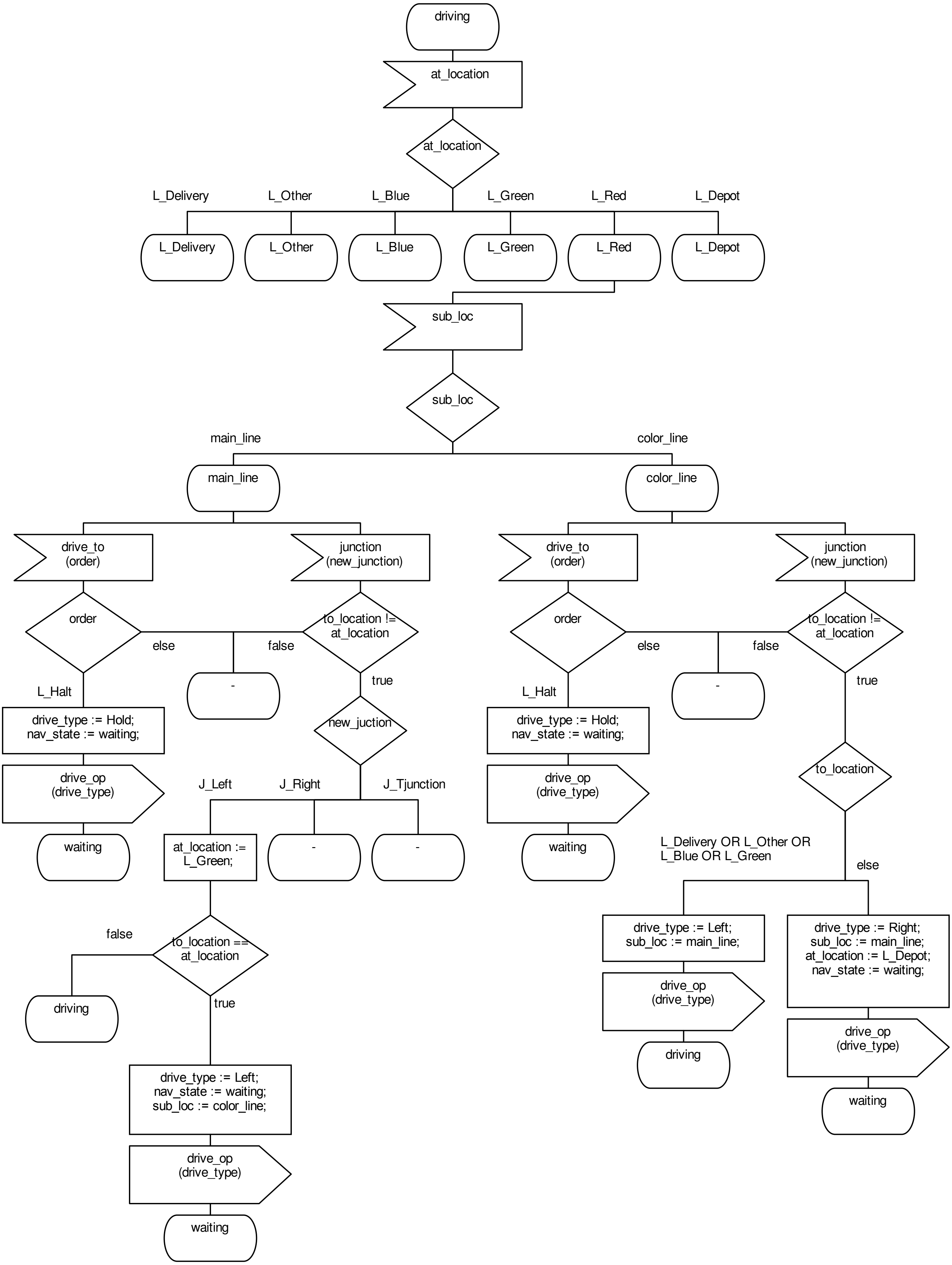


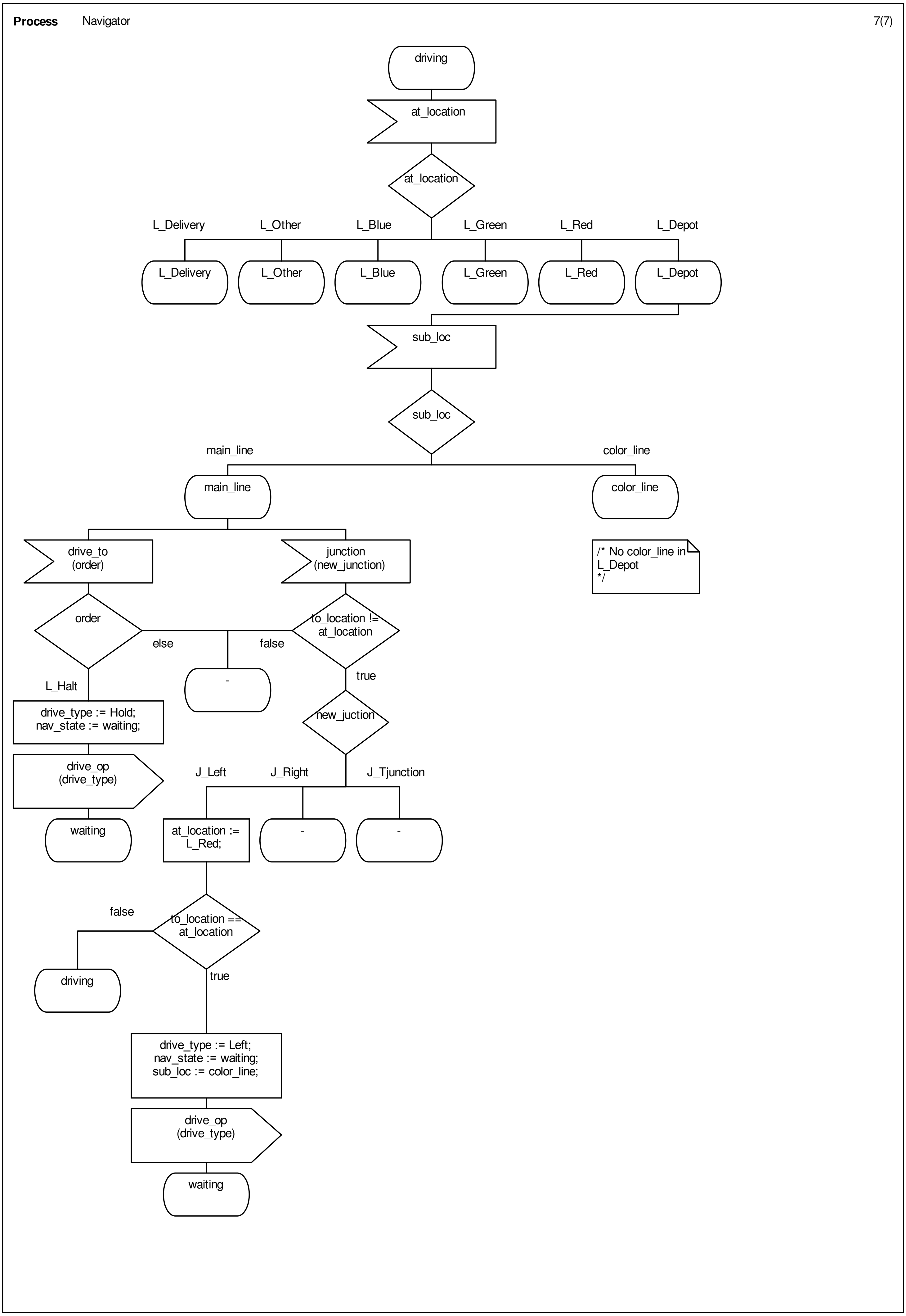
**Process** Navigator 4(7)



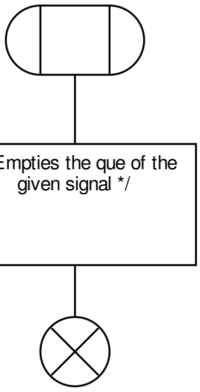

driving
at_location
at_location
L_Delivery
L_Other
L_Blue
L_Green
L_Red
L_Depot
L_Delivery
L_Other
L_Blue
L_Green
L_Red
L_Depot
sub_loc
sub_loc
main_line
color_line
main_line
color_line
drive_to (order)
junction (new_junction)
drive_to (order)
junction (new_junction)
order
else
false
to_location != at_location
order
else
false
to_location != at_location
-
true
-
true
L_Halt
L_Halt
drive_type := Hold; nav_state := waiting;
new_juction
drive_type := Hold; nav_state := waiting;
drive_op (drive_type)
J_Left
J_Right
J_Tjunction
drive_op (drive_type)
to_location
waiting
at_location := L_Blue;
at_location := L_Red;
-
waiting
L_Delivery OR L_Other OR L_Blue
else
false
to_location == at_location
false
to_location == at_location
drive_type := Left; sub_loc := main_line;
drive_type := Right; sub_loc := main_line;
driving
true
true
drive_op (drive_type)
drive_op (drive_type)
nav_state := waiting; sub_loc := main_line; at_location := L_Depot;
drive_type := Right; nav_state := waiting; sub_loc := color_line;
driving
driving
drive_type := Left; nav_state := waiting; sub_loc := color_line;
waiting
drive_op (drive_type)
drive_op (drive_type)
waiting
waiting

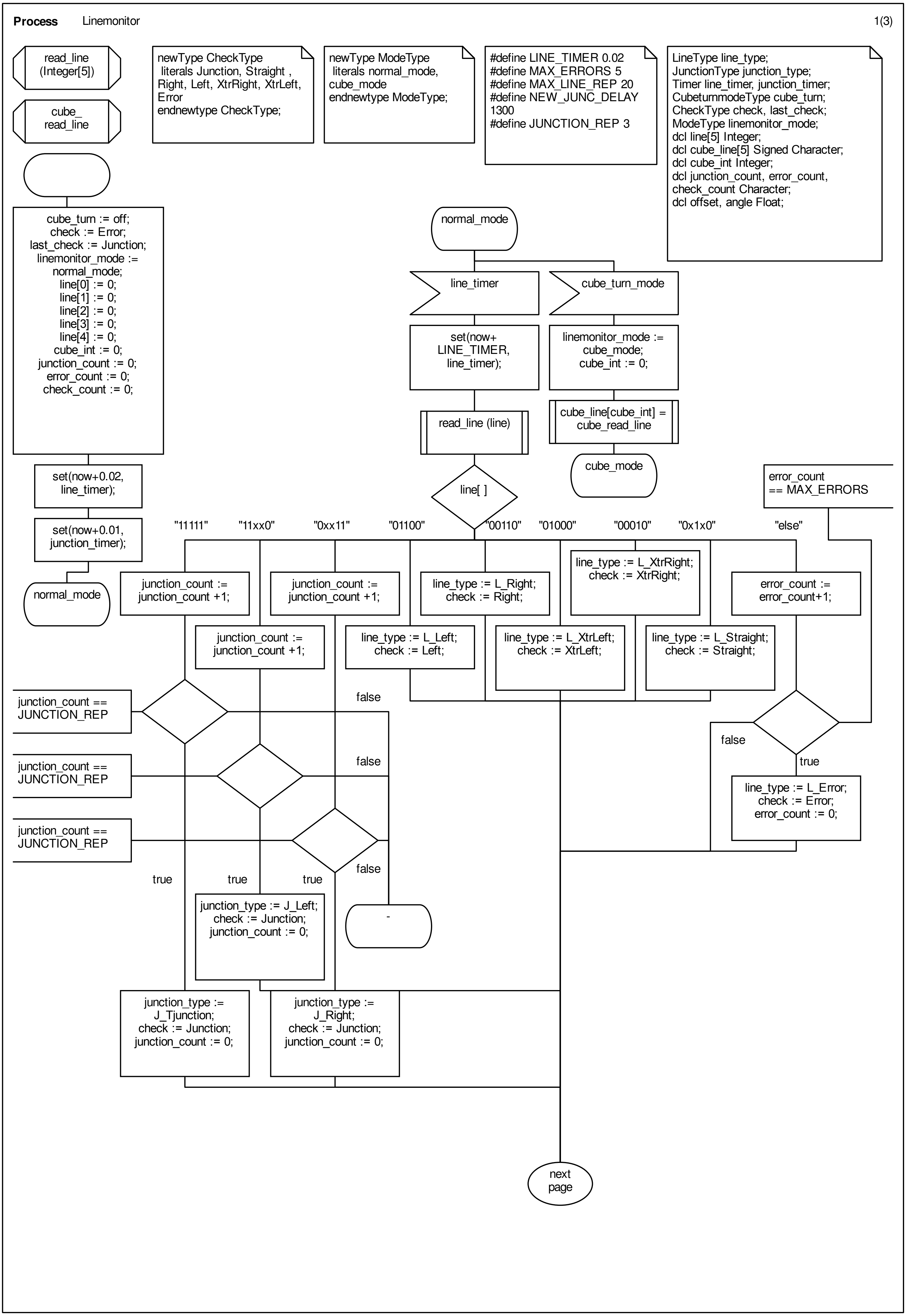
**Process** Navigator 6(7)



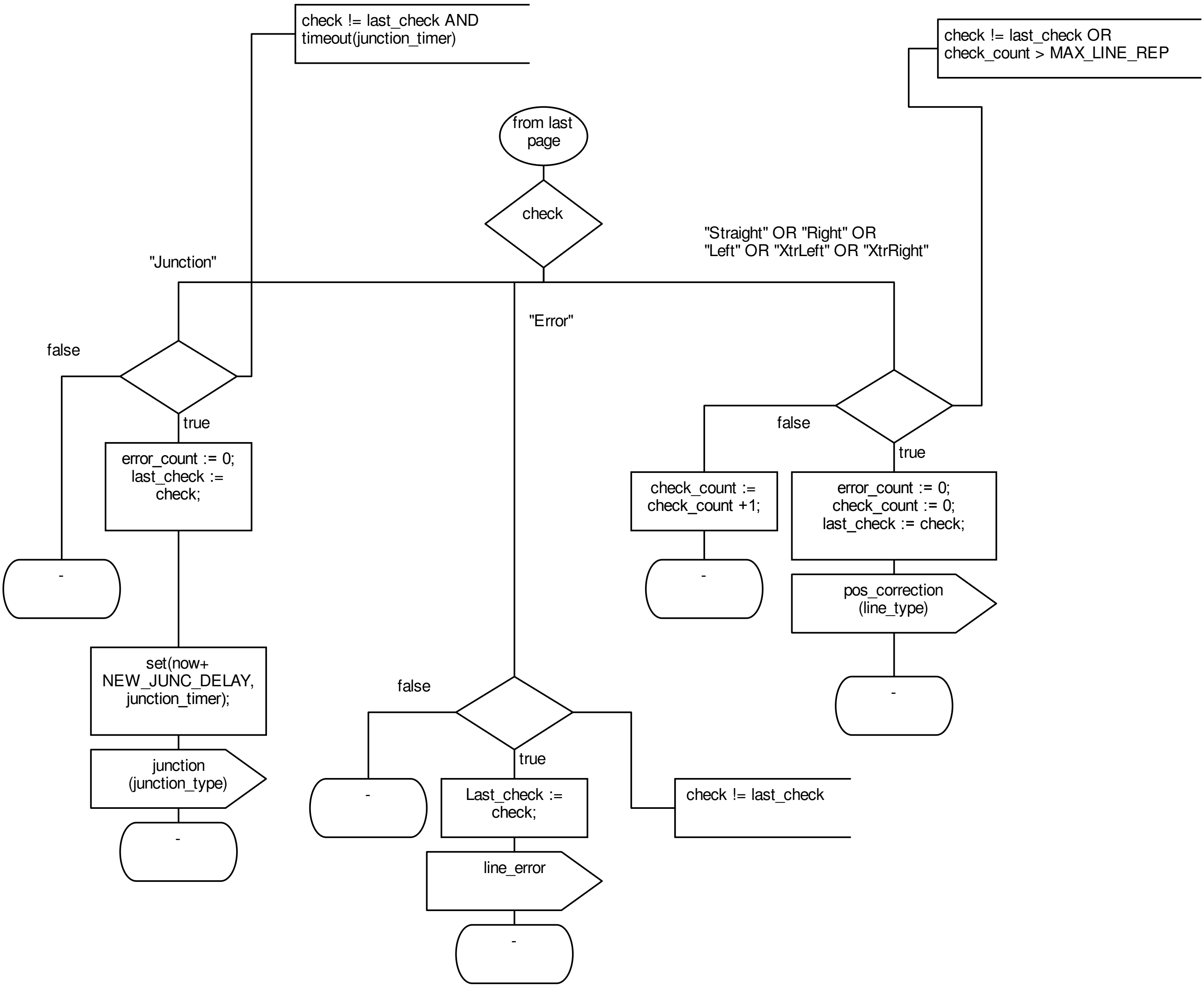

Process Navigator
7(7)
driving
at_location
at_location
L_Delivery
L_Other
L_Blue
L_Green
L_Red
L_Depot
L_Delivery
L_Other
L_Blue
L_Green
L_Red
L_Depot
sub_loc
sub_loc
main_line
color_line
main_line
color_line
drive_to (order)
junction (new_junction)
/* No color_line in L_Depot */
order
else
false
to_location != at_location
L_Halt
-
true
drive_type := Hold; nav_state := waiting;
new_juction
drive_op (drive_type)
J_Left
J_Right
J_Tjunction
waiting
at_location := L_Red;
-
-
false
to_location == at_location
driving
true
drive_type := Left; nav_state := waiting; sub_loc := color_line;
drive_op (drive_type)
waiting

**Procedure** emptySignalQueue 1(1)

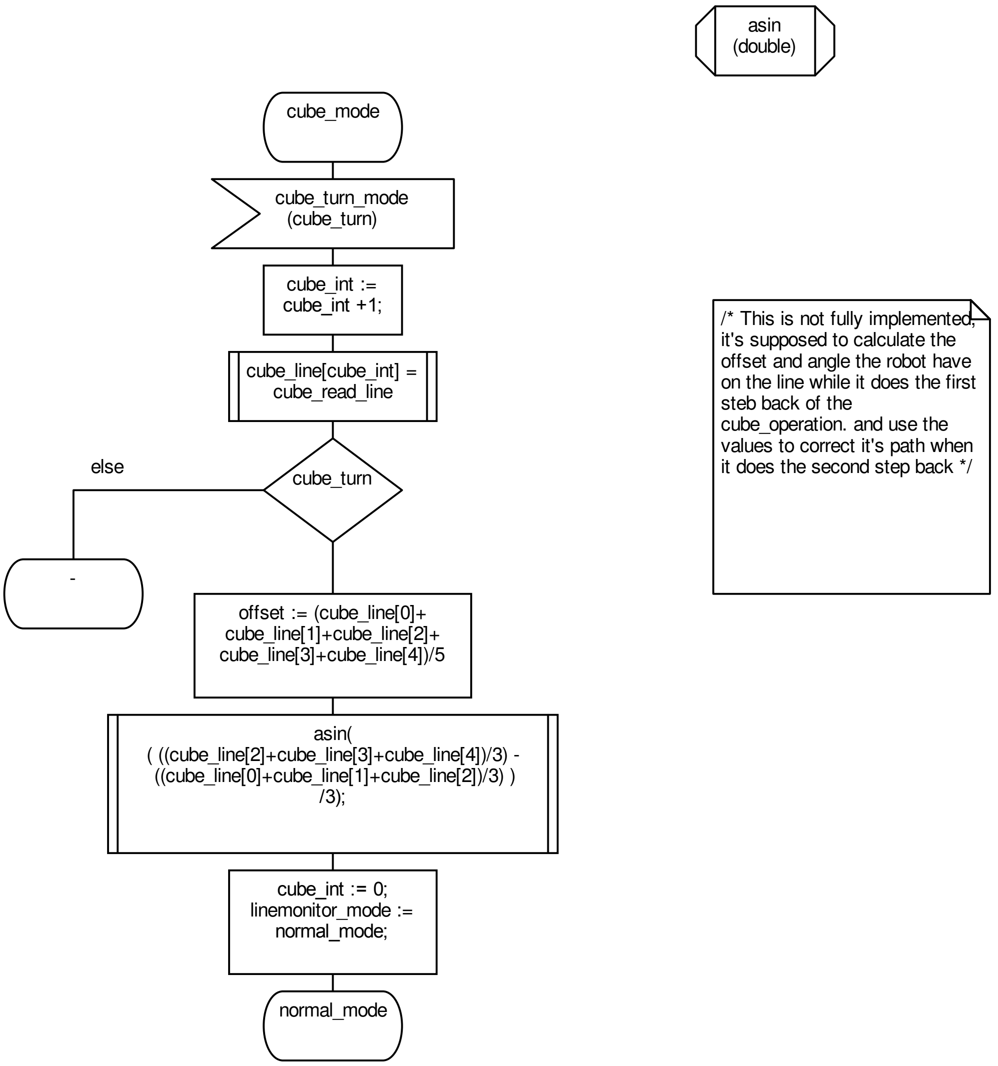



Process Linemonitor
1(3)
read_line (Integer[5])
cube_ read_line
newType CheckType literals Junction, Straight , Right, Left, XtrRight, XtrLeft, Error endnewtype CheckType;
newType ModeType literals normal_mode, cube_mode endnewtype ModeType;
#define LINE_TIMER 0.02
#define MAX_ERRORS 5
#define MAX_LINE_REP 20
#define NEW_JUNC_DELAY 1300
#define JUNCTION_REP 3
LineType line_type;
JunctionType junction_type;
Timer line_timer, junction_timer;
CubeturnmodeType cube_turn;
CheckType check, last_check;
ModeType linemonitor_mode;
dcl line[5] Integer;
dcl cube_line[5] Signed Character;
dcl cube_int Integer;
dcl junction_count, error_count, check_count Character;
dcl offset, angle Float;
cube_turn := off; check := Error; last_check := Junction; linemonitor_mode := normal_mode; line[0] := 0; line[1] := 0; line[2] := 0; line[3] := 0; line[4] := 0; cube_int := 0; junction_count := 0; error_count := 0; check_count := 0;
set(now+0.02, line_timer);
set(now+0.01, junction_timer);
normal_mode
normal_mode
line_timer
cube_turn_mode
set(now+ LINE_TIMER, line_timer);
linemonitor_mode := cube_mode; cube_int := 0;
read_line (line)
cube_line[cube_int] = cube_read_line
cube_mode
line[ ]
error_count == MAX_ERRORS
"11111"
"11xx0"
"0xx11"
"01100"
"00110"
"01000"
"00010"
"0x1x0"
"else"
junction_count := junction_count +1;
junction_count := junction_count +1;
junction_count := junction_count +1;
line_type := L_Right; check := Right;
line_type := L_XtrRight; check := XtrRight;
error_count := error_count+1;
line_type := L_Left; check := Left;
line_type := L_XtrLeft; check := XtrLeft;
line_type := L_Straight; check := Straight;
junction_count == JUNCTION_REP
junction_count == JUNCTION_REP
junction_count == JUNCTION_REP
false
false
false
false
true
true
true
true
line_type := L_Error; check := Error; error_count := 0;
junction_type := J_Left; check := Junction; junction_count := 0;
-
junction_type := J_Tjunction; check := Junction; junction_count := 0;
junction_type := J_Right; check := Junction; junction_count := 0;
next page

**Process** Linemonitor

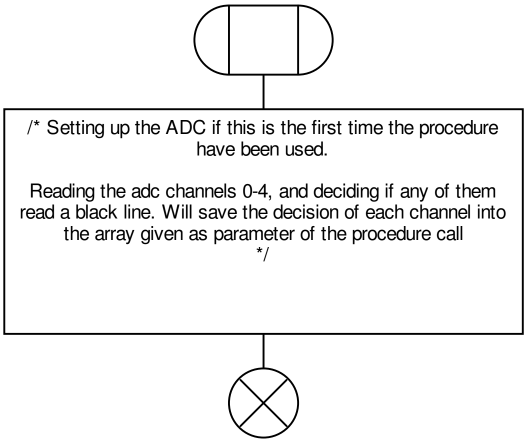


**Process** Linemonitor 3(3)



/* This is not fully implemented, it's supposed to calculate the offset and angle the robot have on the line while it does the first steb back of the cube_operation. and use the values to correct it's path when it does the second step back */

**Procedure** read_line 1(1)

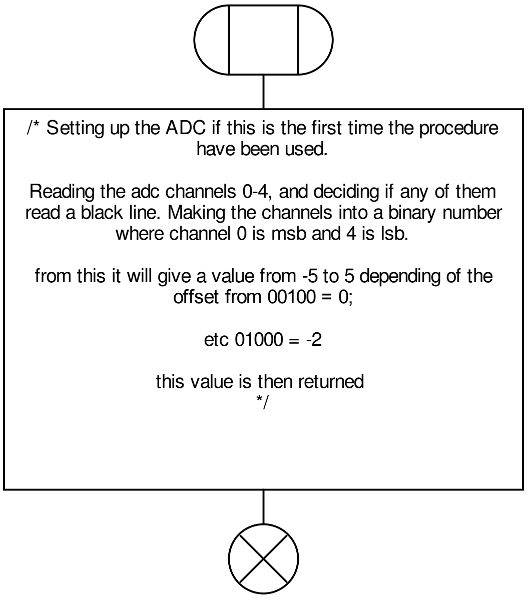


/* Setting up the ADC if this is the first time the procedure have been used.

Reading the adc channels 0-4, and deciding if any of them read a black line. Will save the decision of each channel into the array given as parameter of the procedure call
*/

**Procedure** cube_
read_line

1(1)



/* Setting up the ADC if this is the first time the procedure have been used.

Reading the adc channels 0-4, and deciding if any of them read a black line. Making the channels into a binary number where channel 0 is msb and 4 is lsb.

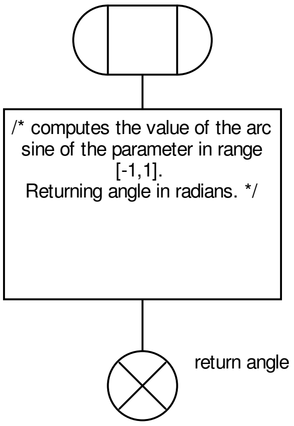
from this it will give a value from -5 to 5 depending of the offset from 00100 = 0;

etc 01000 = -2

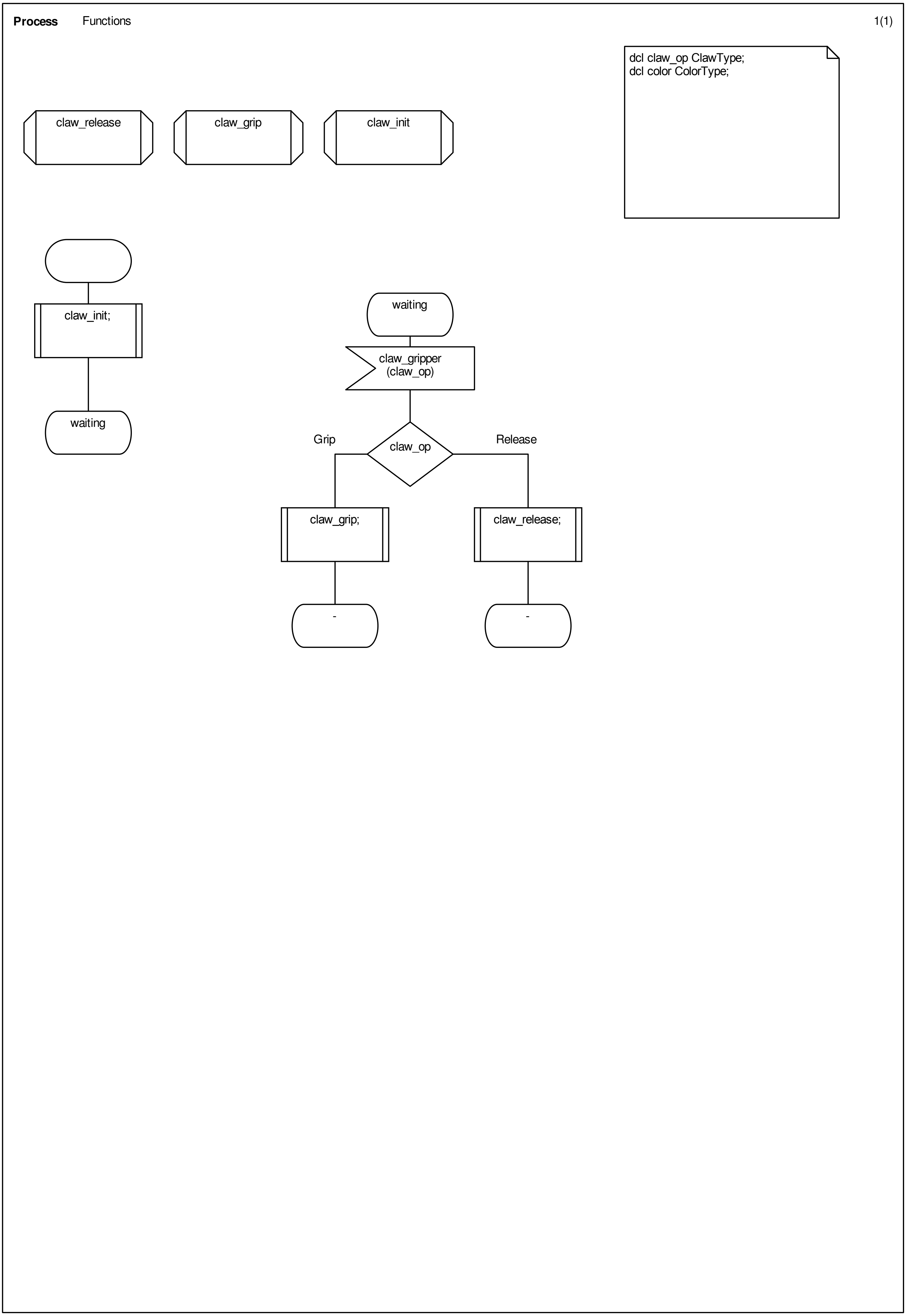
this value is then returned
*/

**Procedure** asin

1(1)

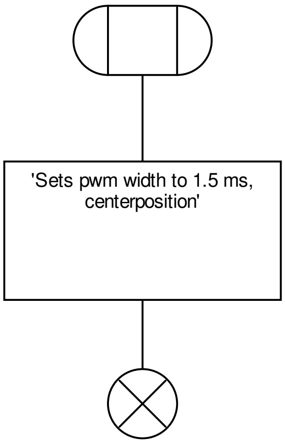


/* computes the value of the arc sine of the parameter in range [-1,1].
Returning angle in radians. */

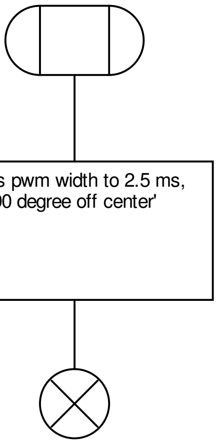


```
dcl claw_op ClawType;
dcl color ColorType;
```

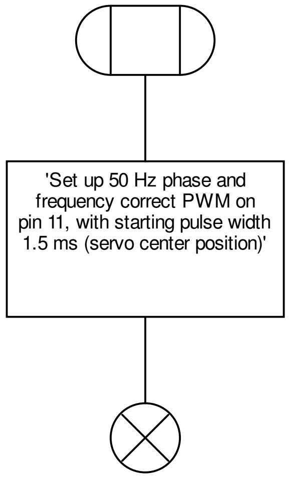
**Procedure** claw_release 1(1)



**Procedure** claw_grip



**Procedure** claw_init 1(1)


'Set up 50 Hz phase and frequency correct PWM on pin 11, with starting pulse width 1.5 ms (servo center position)'

**Process** Navigator_drive 1(4)

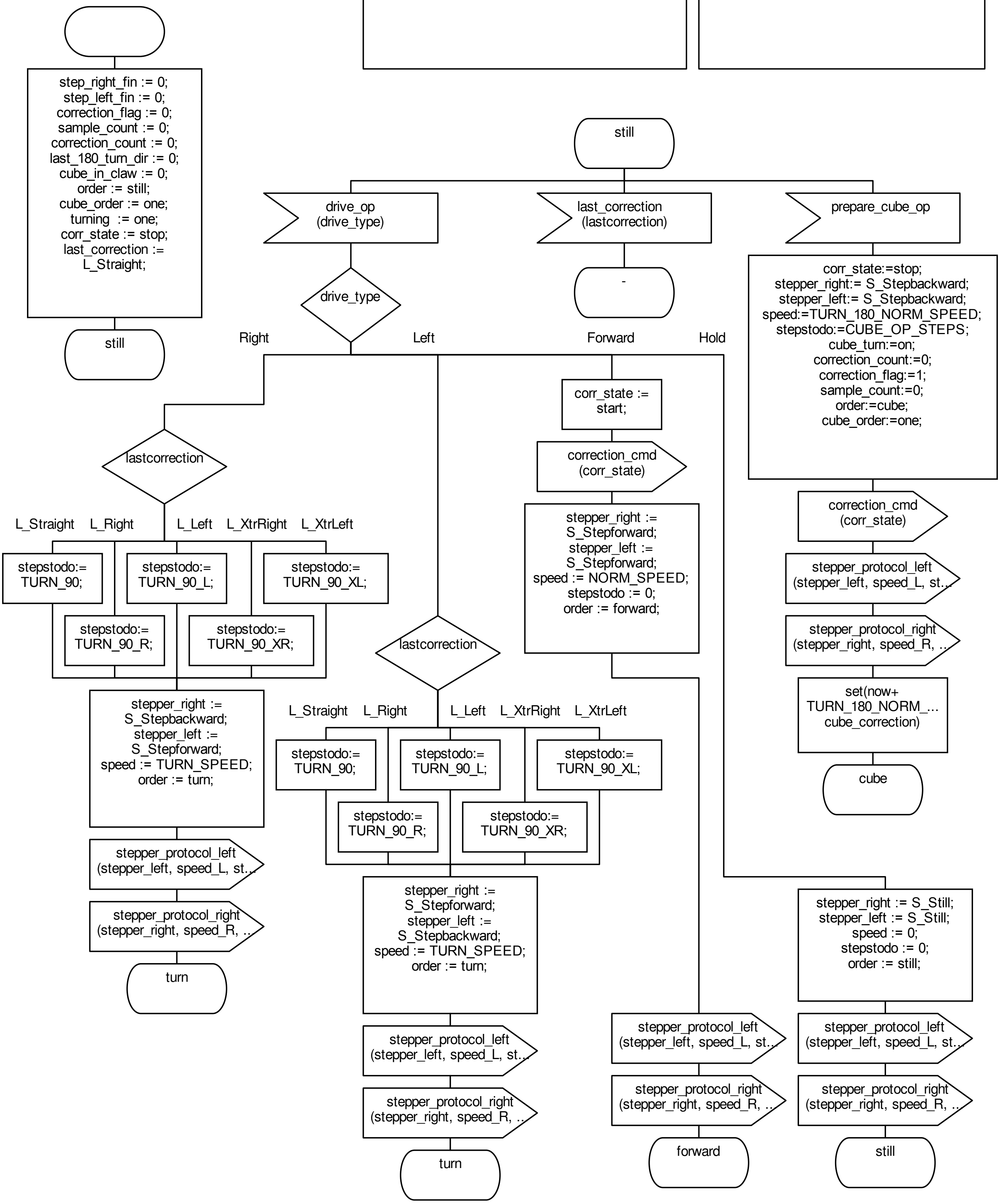
```
newType MainorderType
 literals still, forward, cube, turn
endnewtype MainorderType;
```

```
newType OrderType
 literals one, two, three, four, five
endnewtype OrderType;
```

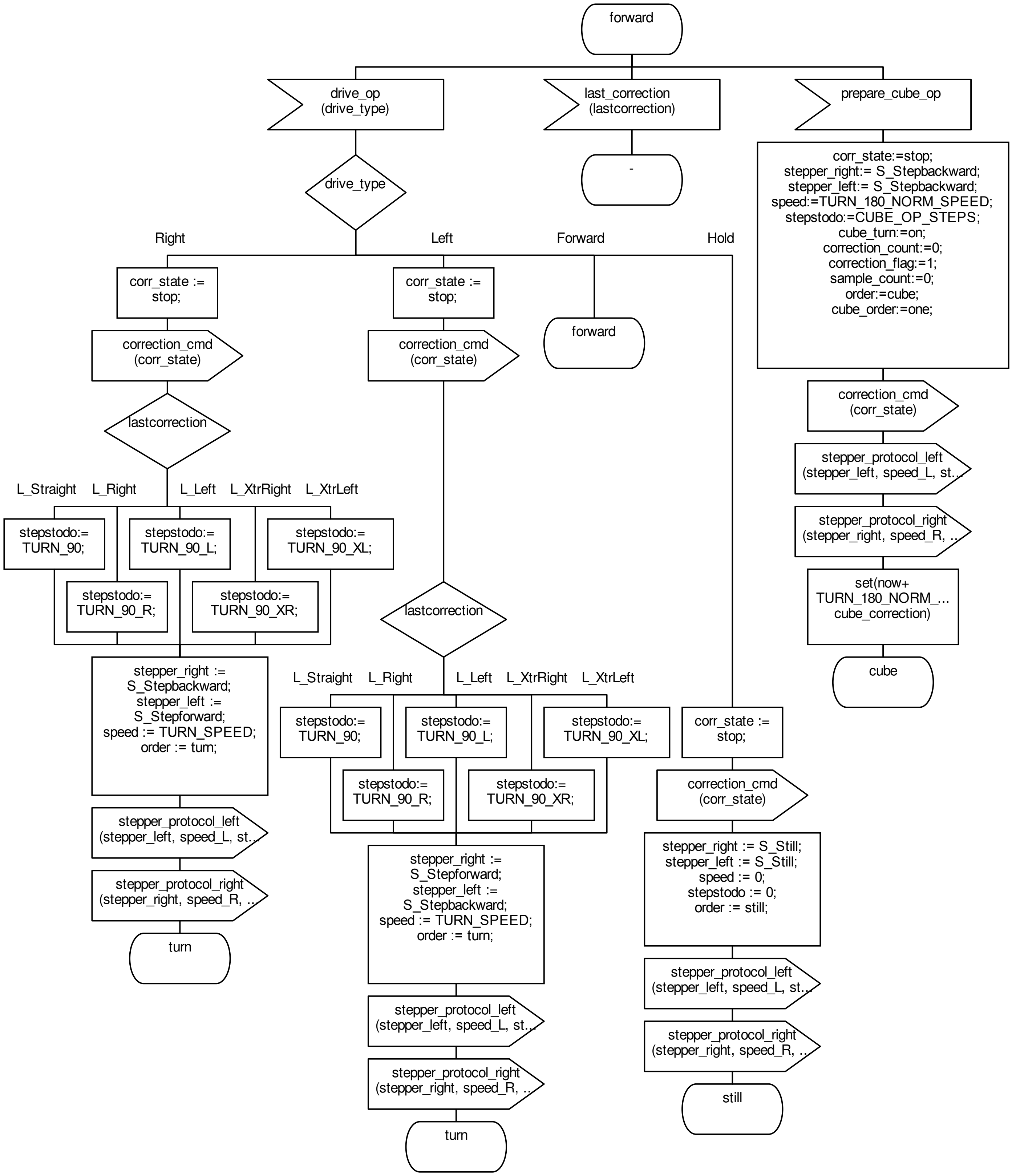
```
#define NORM_SPEED 8
#define TURN_SPEED 14
#define TURN_180_NORM_SPEED 16
#define TURN_90 62
#define TURN_90_L TURN_90+2
#define TURN_90_R TURN_90-2
#define TURN_90_XL TURN_90_R+4
#define TURN_90_XR TURN_90_L-4
#define TURN_180 127
#define TURN_180_L TURN_180-3
#define TURN_180_R TURN_180+3
#define TURN_180_XL TURN_180_L-8
#define TURN_180_XR TURN_180_R+8
#define CUBE_OP_STEPS 65
#define WAIT_DURING_TURN_DELAY 2500
```

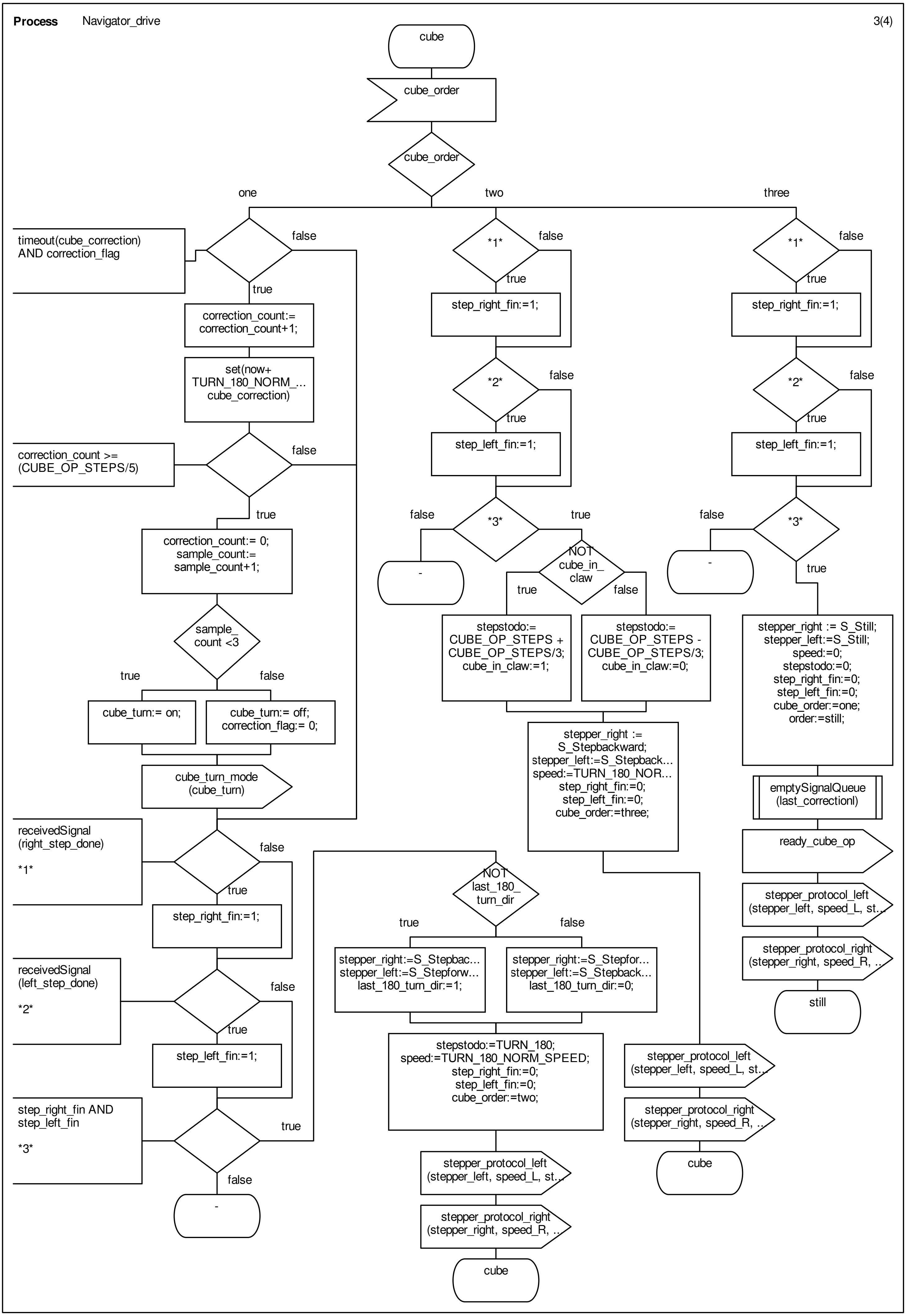
```
DriveType drive_type;
LineType lastcorrection;
StepperType stepper_right, stepper_left;
CubeturnmodeType cube_turn;
CorrectioncmdType corr_state;
MainorderType order;
OrderType cube_order, turning;
Timer turningtimer, cube_correction;
dcl speed, correction_flag, sample_count,
correction_count, last_180_turn_dir,
cube_in_claw Character;
dcl stepstodo, step_right_fin, step_left_fin
Integer;
```

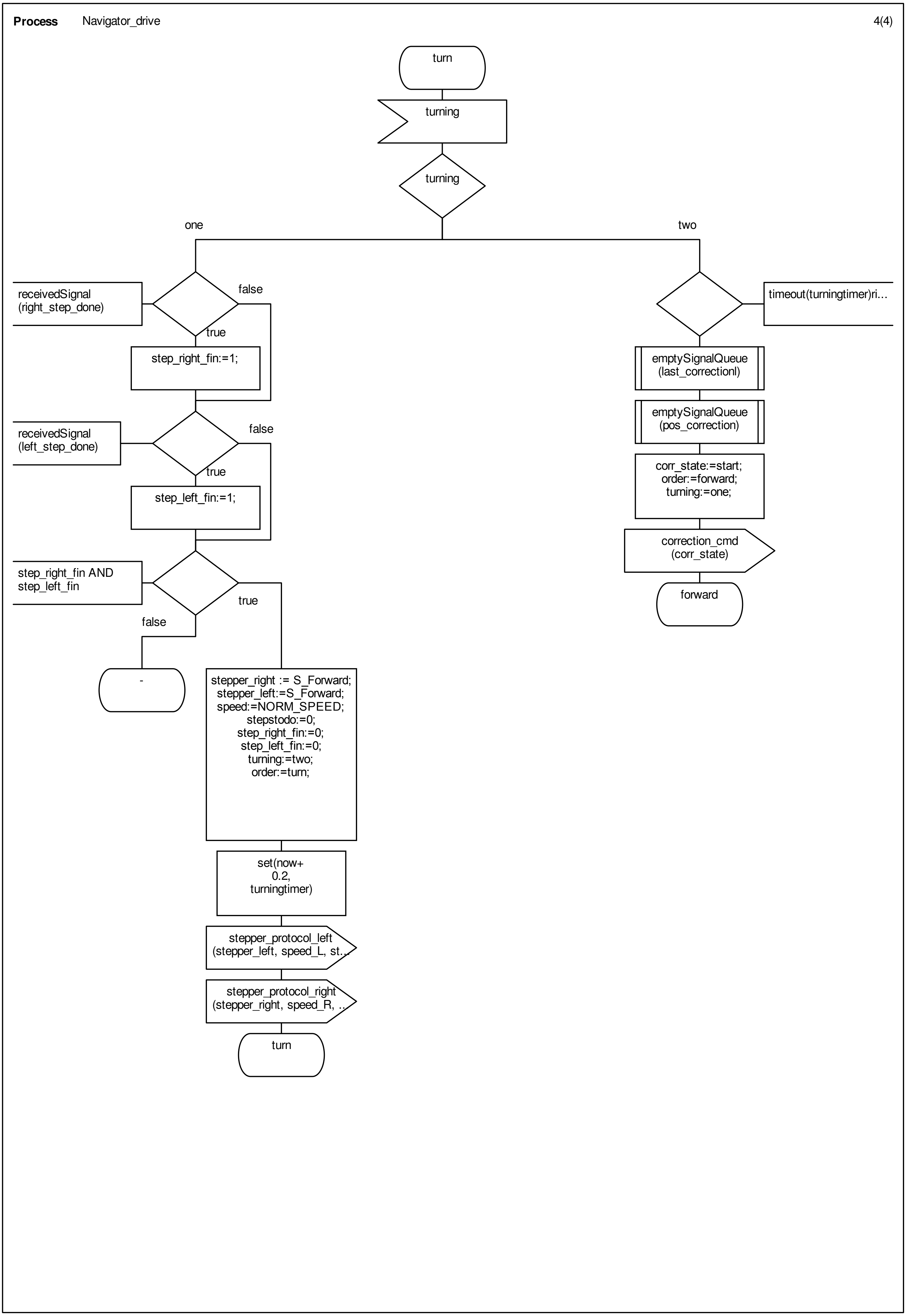
emptySignalQueue(signal)



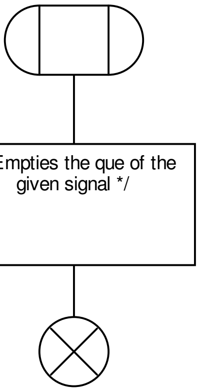
**Process** Navigator_drive 2(4)




Process Navigator_drive
3(4)
cube
cube_order
cube_order
one
two
three
timeout(cube_correction)
AND correction_flag
false
true
correction_count:=
correction_count+1;
set(now+
TURN_180_NORM_...
cube_correction)
correction_count >=
(CUBE_OP_STEPS/5)
false
true
correction_count:= 0;
sample_count:=
sample_count+1;
sample_
count <3
true
false
cube_turn:= on;
cube_turn:= off;
correction_flag:= 0;
cube_turn_mode
(cube_turn)
receivedSignal
(right_step_done)
*1*
false
true
step_right_fin:=1;
receivedSignal
(left_step_done)
*2*
false
true
step_left_fin:=1;
step_right_fin AND
step_left_fin
*3*
true
false
-
*1*
false
true
step_right_fin:=1;
*2*
false
true
step_left_fin:=1;
false
*3*
true
-
NOT
cube_in_
claw
true
false
stepstodo:=
CUBE_OP_STEPS +
CUBE_OP_STEPS/3;
cube_in_claw:=1;
stepstodo:=
CUBE_OP_STEPS -
CUBE_OP_STEPS/3;
cube_in_claw:=0;
stepper_right :=
S_Stepbackward;
stepper_left:=S_Stepback...
speed:=TURN_180_NOR...
step_right_fin:=0;
step_left_fin:=0;
cube_order:=three;
NOT
last_180_
turn_dir
true
false
stepper_right:=S_Stepbac...
stepper_left:=S_Stepforw...
last_180_turn_dir:=1;
stepper_right:=S_Stepfor...
stepper_left:=S_Stepback...
last_180_turn_dir:=0;
stepstodo:=TURN_180;
speed:=TURN_180_NORM_SPEED;
step_right_fin:=0;
step_left_fin:=0;
cube_order:=two;
stepper_protocol_left
(stepper_left, speed_L, st...
stepper_protocol_right
(stepper_right, speed_R, ...
cube
stepper_protocol_left
(stepper_left, speed_L, st...
stepper_protocol_right
(stepper_right, speed_R, ...
cube
*1*
false
true
step_right_fin:=1;
*2*
false
true
step_left_fin:=1;
false
*3*
-
true
stepper_right := S_Still;
stepper_left:=S_Still;
speed:=0;
stepstodo:=0;
step_right_fin:=0;
step_left_fin:=0;
cube_order:=one;
order:=still;
emptySignalQueue
(last_correctionl)
ready_cube_op
stepper_protocol_left
(stepper_left, speed_L, st...
stepper_protocol_right
(stepper_right, speed_R, ...
still


Process
Navigator_drive
4(4)
turn
turning
turning
one
two
receivedSignal
(right_step_done)
false
true
step_right_fin:=1;
receivedSignal
(left_step_done)
false
true
step_left_fin:=1;
step_right_fin AND
step_left_fin
true
false
-
stepper_right := S_Forward;
stepper_left:=S_Forward;
speed:=NORM_SPEED;
stepstodo:=0;
step_right_fin:=0;
step_left_fin:=0;
turning:=two;
order:=turn;
set(now+
0.2,
turningtimer)
stepper_protocol_left
(stepper_left, speed_L, st...
stepper_protocol_right
(stepper_right, speed_R, ...
turn
timeout(turningtimer)ri...
emptySignalQueue
(last_correctionl)
emptySignalQueue
(pos_correction)
corr_state:=start;
order:=forward;
turning:=one;
correction_cmd
(corr_state)
forward

**Procedure** emptySignalQueue 1(1)



**Process** Correction

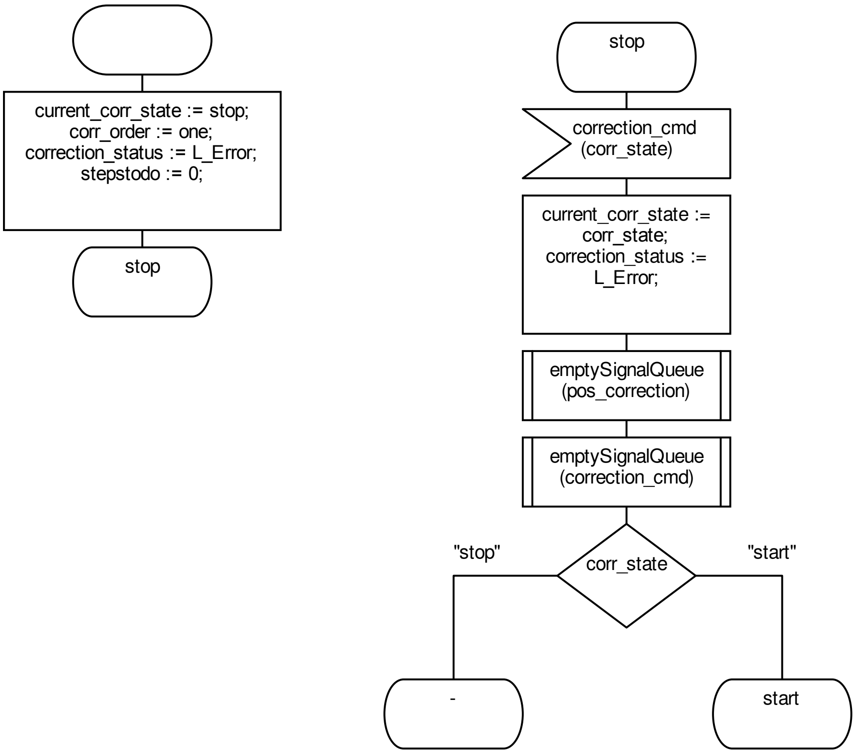
1(8)

emptySignalQueue(signal)

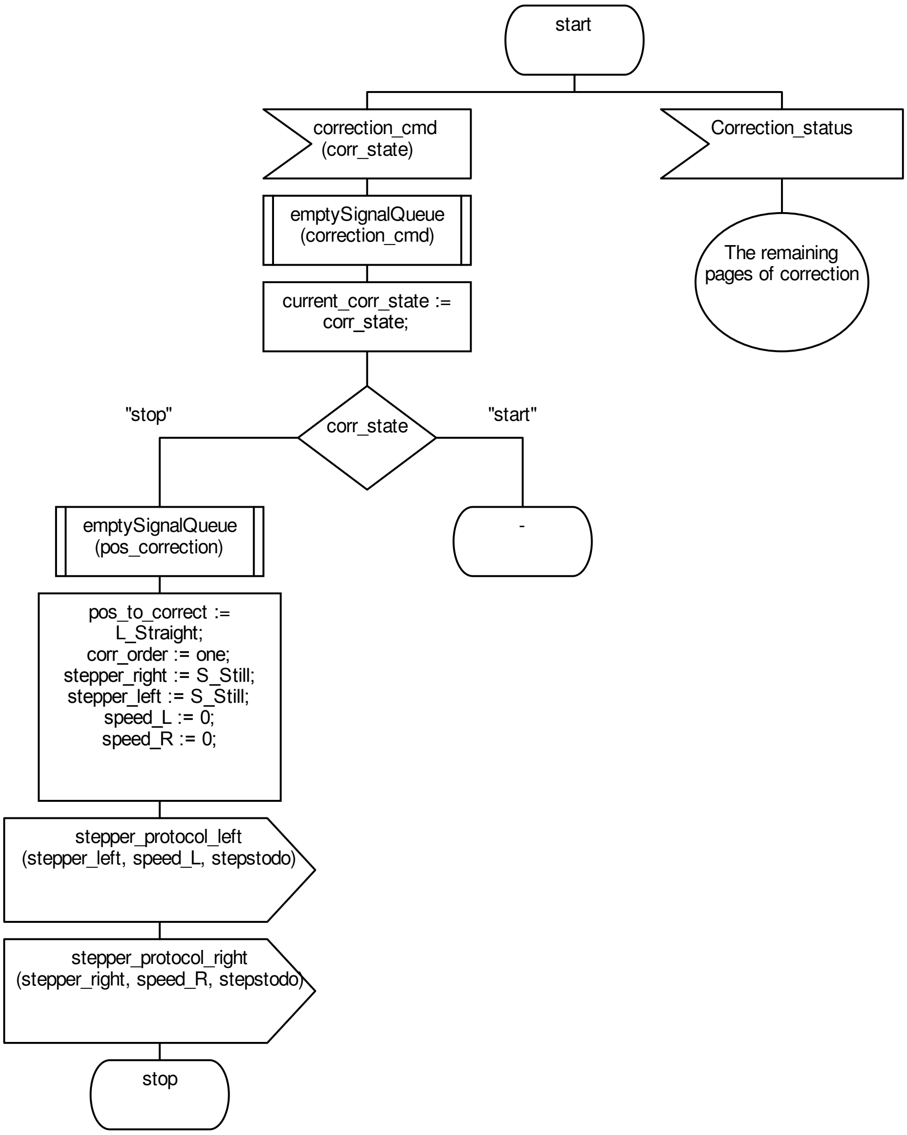
```
#define TIMERDUR_1 0.2
#define TIMERDUR_2 0.133
#define TIMERDUR_LONG_1
TIMERDUR_1+0.2
#define TIMERDUR_LONG_2
TIMERDUR_2+0.133

#define NORM_SPEED 7
#define SOFT_CORR_SPEED_FAST 7
#define SOFT_CORR_SPEED_SLOW 11
#define HARD_CORR_SPEED_FAST 7
#define HARD_CORR_SPEED_SLOW 15
```

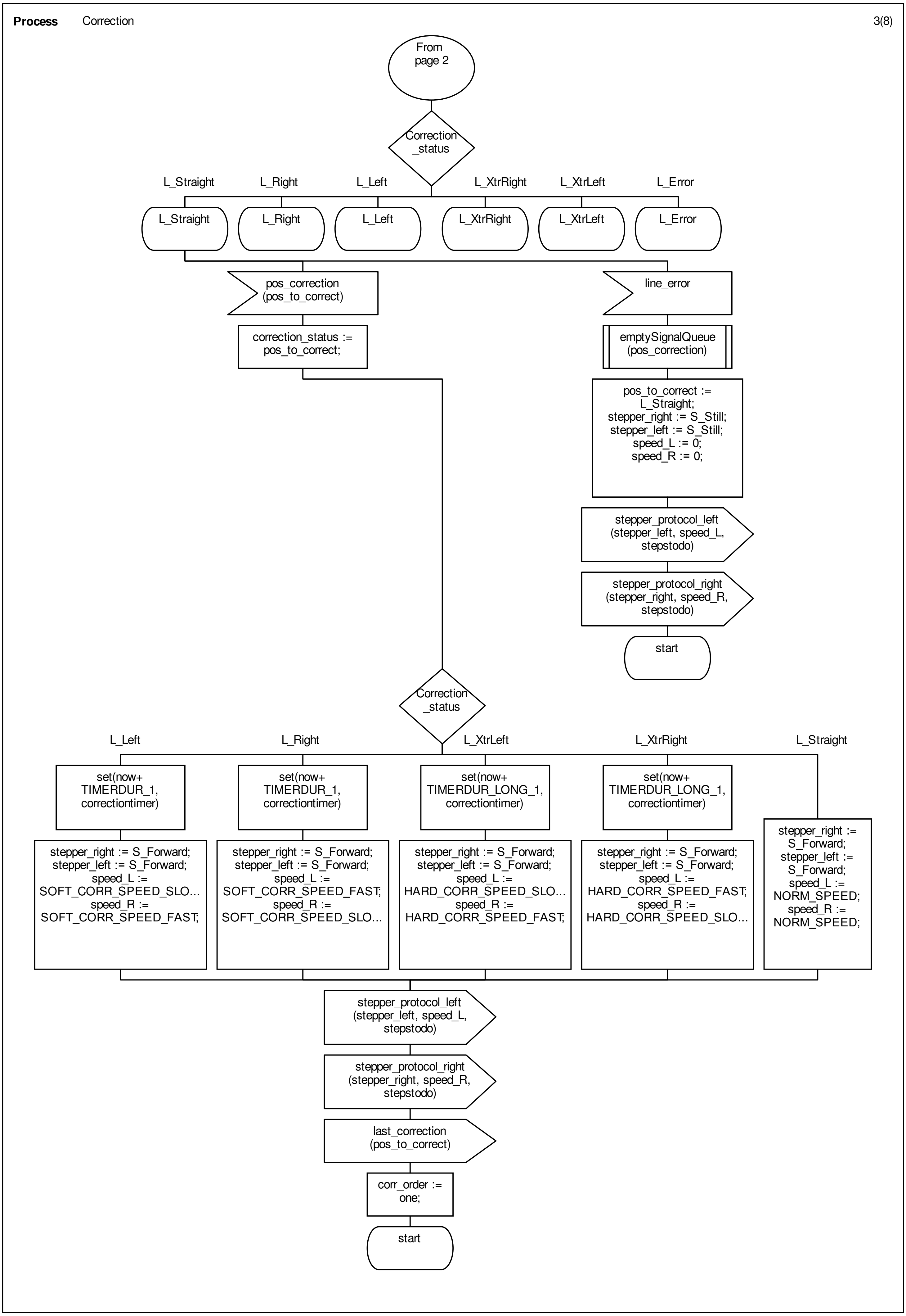
```
newType OrderType
 literals one, two, three, four
endnewtype OrderType;
```

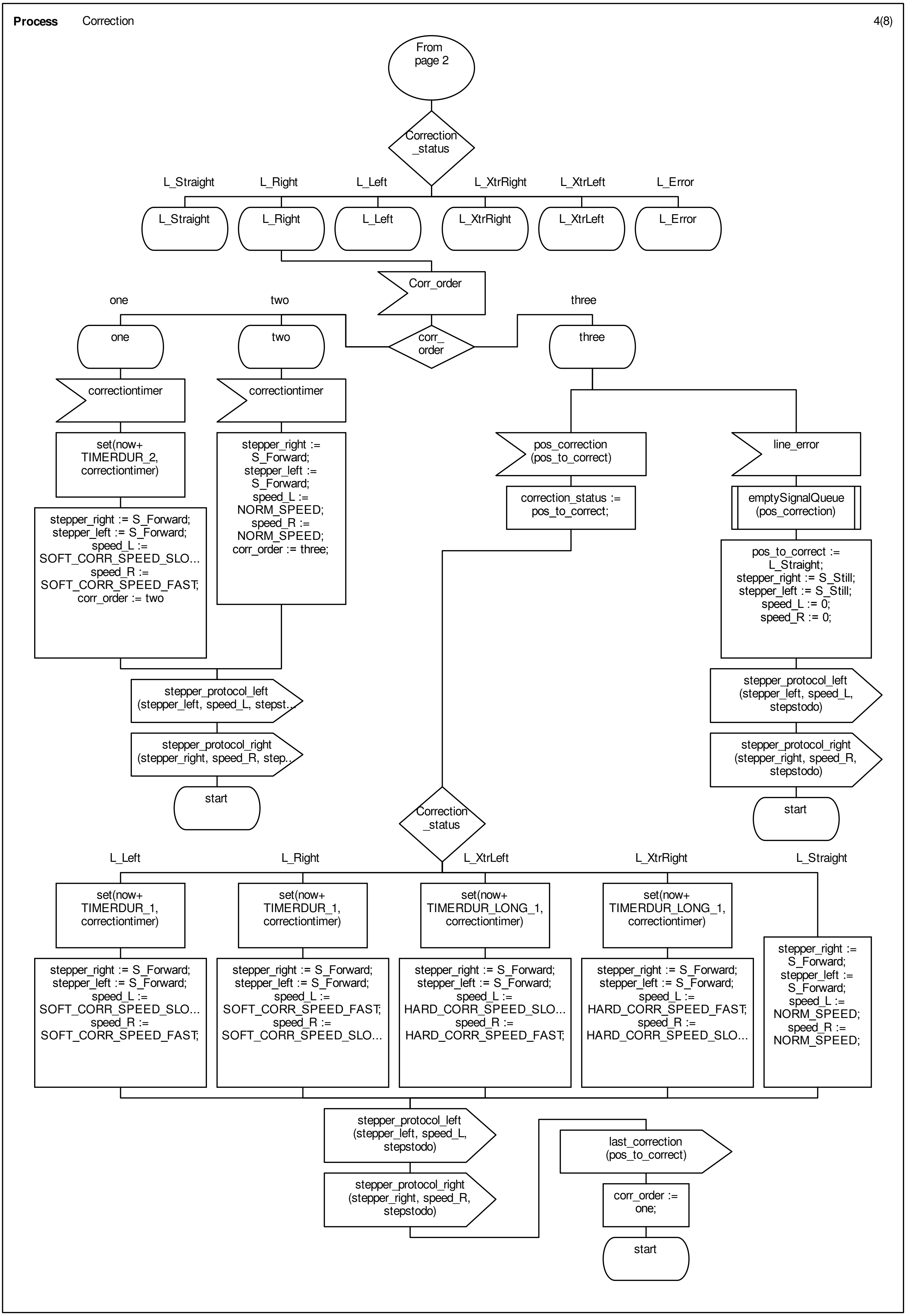
```
dcl speed_L, speed_R Character;
dcl stepstodo Integer;
LineType pos_to_correct,
correction_status;
StepperType stepper_right,
stepper_left;
OrderType corr_order;
CorrectioncmdType corr_state,
current_corr_state;
Timer correctiontimer;
```

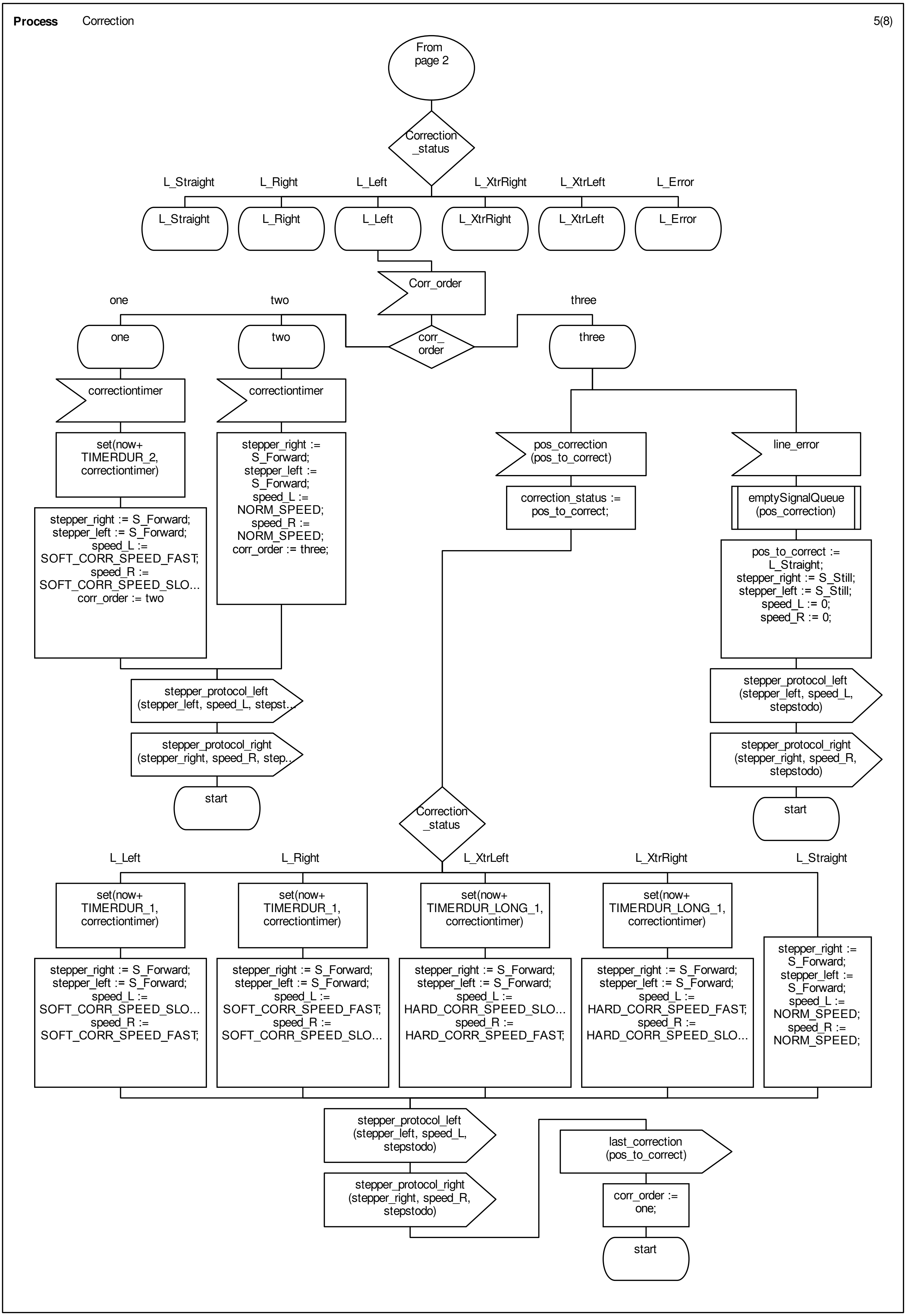


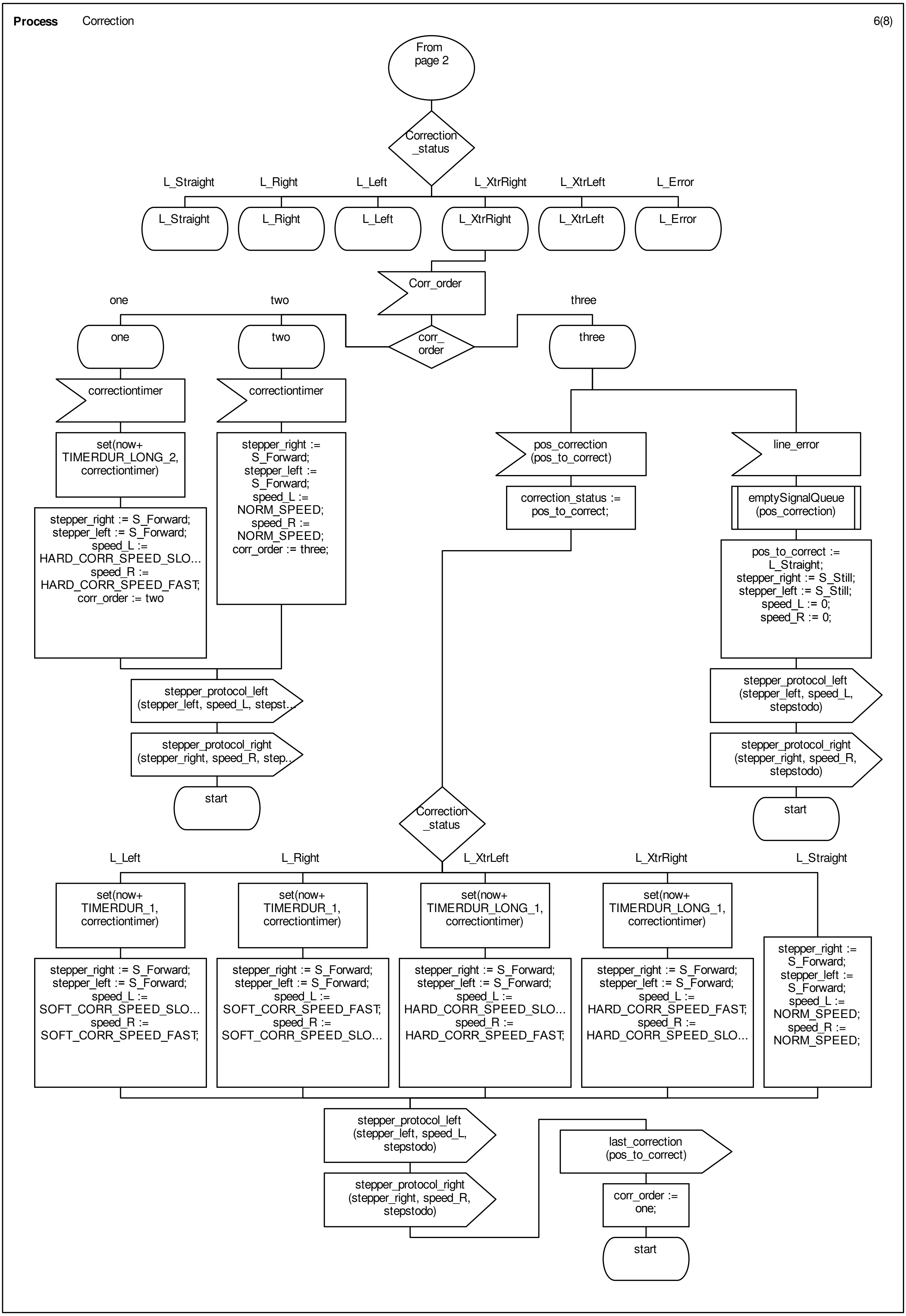
**Process** Correction 2(8)



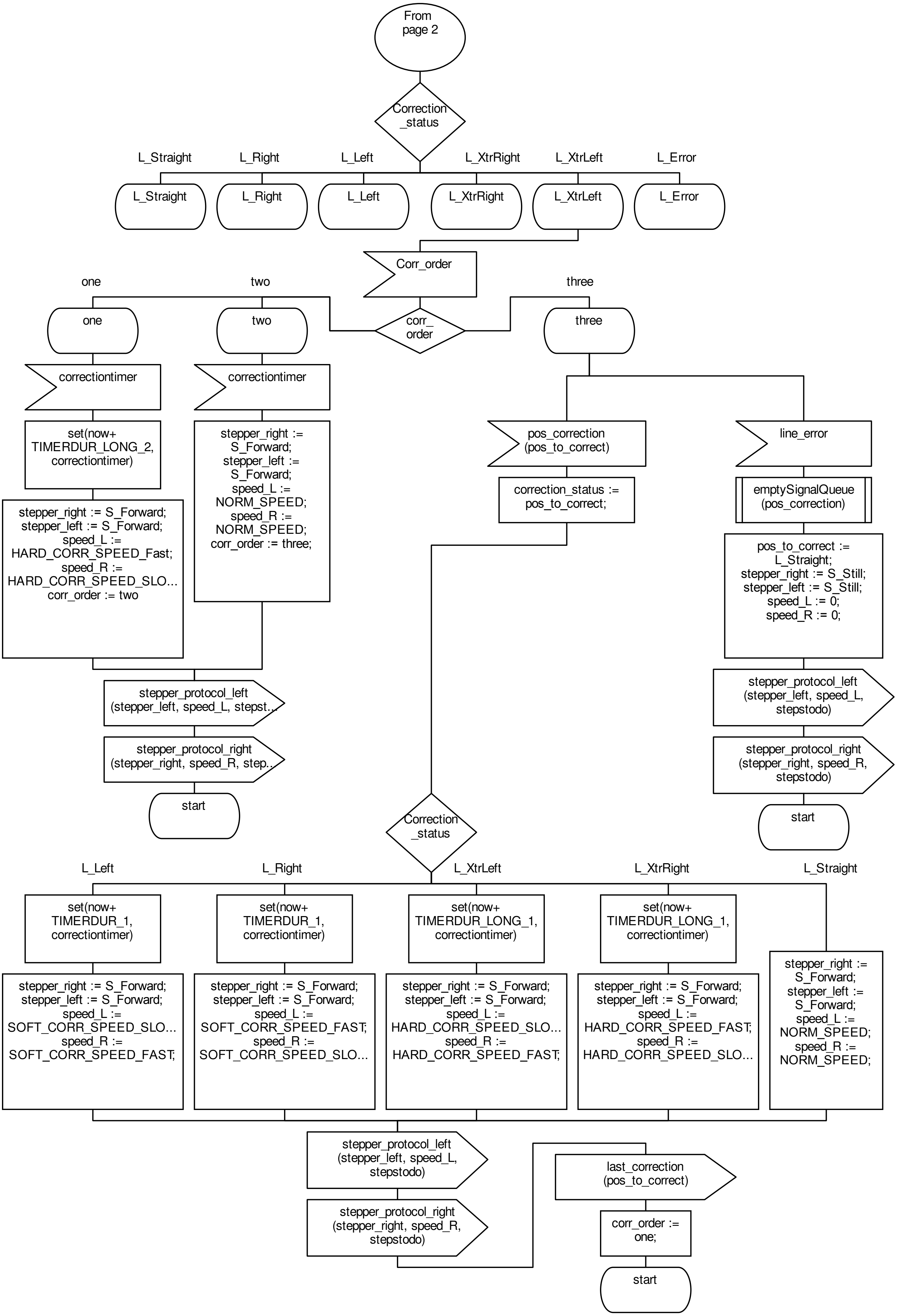

Process
Correction
3(8)
From page 2
Correction_status
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
pos_correction (pos_to_correct)
correction_status := pos_to_correct;
line_error
emptySignalQueue (pos_correction)
pos_to_correct := L_Straight; stepper_right := S_Still; stepper_left := S_Still; speed_L := 0; speed_R := 0;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
start
Correction_status
L_Left
L_Right
L_XtrLeft
L_XtrRight
L_Straight
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_FAST; speed_R := HARD_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
last_correction (pos_to_correct)
corr_order := one;
start


Process
Correction
4(8)
From page 2
Correction _status
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
Corr_order
one
two
three
corr_ order
one
two
three
correctiontimer
correctiontimer
set(now+ TIMERDUR_2, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST; corr_order := two
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED; corr_order := three;
stepper_protocol_left (stepper_left, speed_L, stepst...
stepper_protocol_right (stepper_right, speed_R, step...
start
pos_correction (pos_to_correct)
correction_status := pos_to_correct;
line_error
emptySignalQueue (pos_correction)
pos_to_correct := L_Straight; stepper_right := S_Still; stepper_left := S_Still; speed_L := 0; speed_R := 0;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
start
Correction _status
L_Left
L_Right
L_XtrLeft
L_XtrRight
L_Straight
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_FAST; speed_R := HARD_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
last_correction (pos_to_correct)
corr_order := one;
start

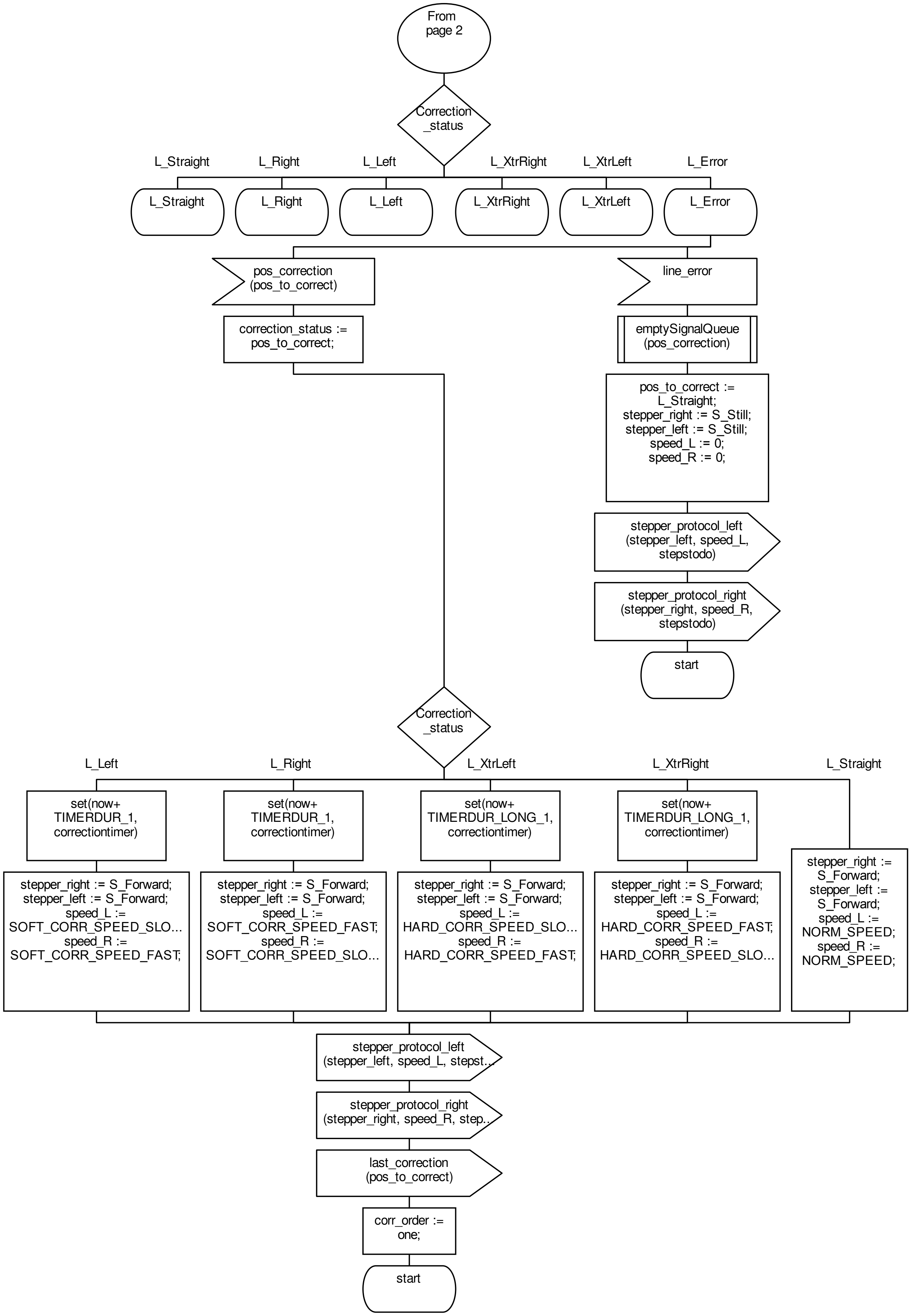

Process
Correction
5(8)
From page 2
Correction _status
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
Corr_order
one
two
three
corr_ order
one
two
three
correctiontimer
correctiontimer
set(now+ TIMERDUR_2, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO... corr_order := two
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED; corr_order := three;
stepper_protocol_left (stepper_left, speed_L, stepst...
stepper_protocol_right (stepper_right, speed_R, step...
start
pos_correction (pos_to_correct)
correction_status := pos_to_correct;
line_error
emptySignalQueue (pos_correction)
pos_to_correct := L_Straight; stepper_right := S_Still; stepper_left := S_Still; speed_L := 0; speed_R := 0;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
start
Correction _status
L_Left
L_Right
L_XtrLeft
L_XtrRight
L_Straight
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_FAST; speed_R := HARD_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
last_correction (pos_to_correct)
corr_order := one;
start


Process
Correction
6(8)
From page 2
Correction _status
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
Corr_order
one
two
three
corr_ order
one
two
three
correctiontimer
correctiontimer
set(now+ TIMERDUR_LONG_2, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST; corr_order := two
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED; corr_order := three;
stepper_protocol_left (stepper_left, speed_L, stepst...
stepper_protocol_right (stepper_right, speed_R, step...
start
pos_correction (pos_to_correct)
correction_status := pos_to_correct;
line_error
emptySignalQueue (pos_correction)
pos_to_correct := L_Straight; stepper_right := S_Still; stepper_left := S_Still; speed_L := 0; speed_R := 0;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
start
Correction _status
L_Left
L_Right
L_XtrLeft
L_XtrRight
L_Straight
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_FAST; speed_R := HARD_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
last_correction (pos_to_correct)
corr_order := one;
start

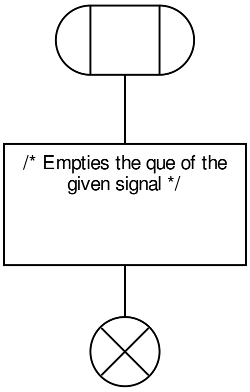
**Process** Correction 7(8)

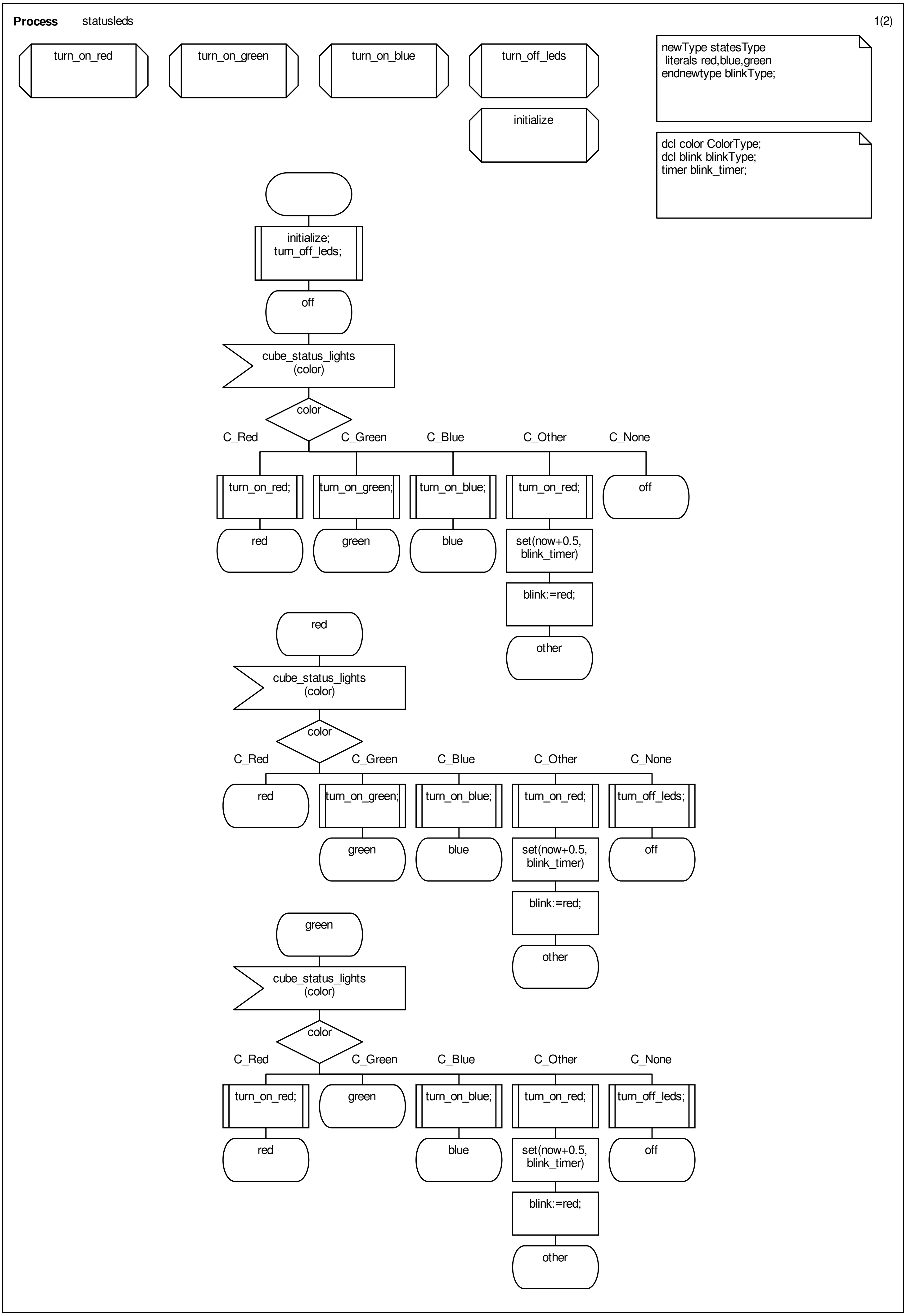

From page 2
Correction _status
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
Corr_order
one
two
three
corr_ order
one
two
three
correctiontimer
correctiontimer
set(now+ TIMERDUR_LONG_2, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_Fast; speed_R := HARD_CORR_SPEED_SLO... corr_order := two
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED; corr_order := three;
pos_correction (pos_to_correct)
correction_status := pos_to_correct;
line_error
emptySignalQueue (pos_correction)
pos_to_correct := L_Straight; stepper_right := S_Still; stepper_left := S_Still; speed_L := 0; speed_R := 0;
stepper_protocol_left (stepper_left, speed_L, stepst...
stepper_protocol_right (stepper_right, speed_R, step...
start
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
start
Correction _status
L_Left
L_Right
L_XtrLeft
L_XtrRight
L_Straight
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_FAST; speed_R := HARD_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
last_correction (pos_to_correct)
corr_order := one;
start

**Process** Correction 8(8)

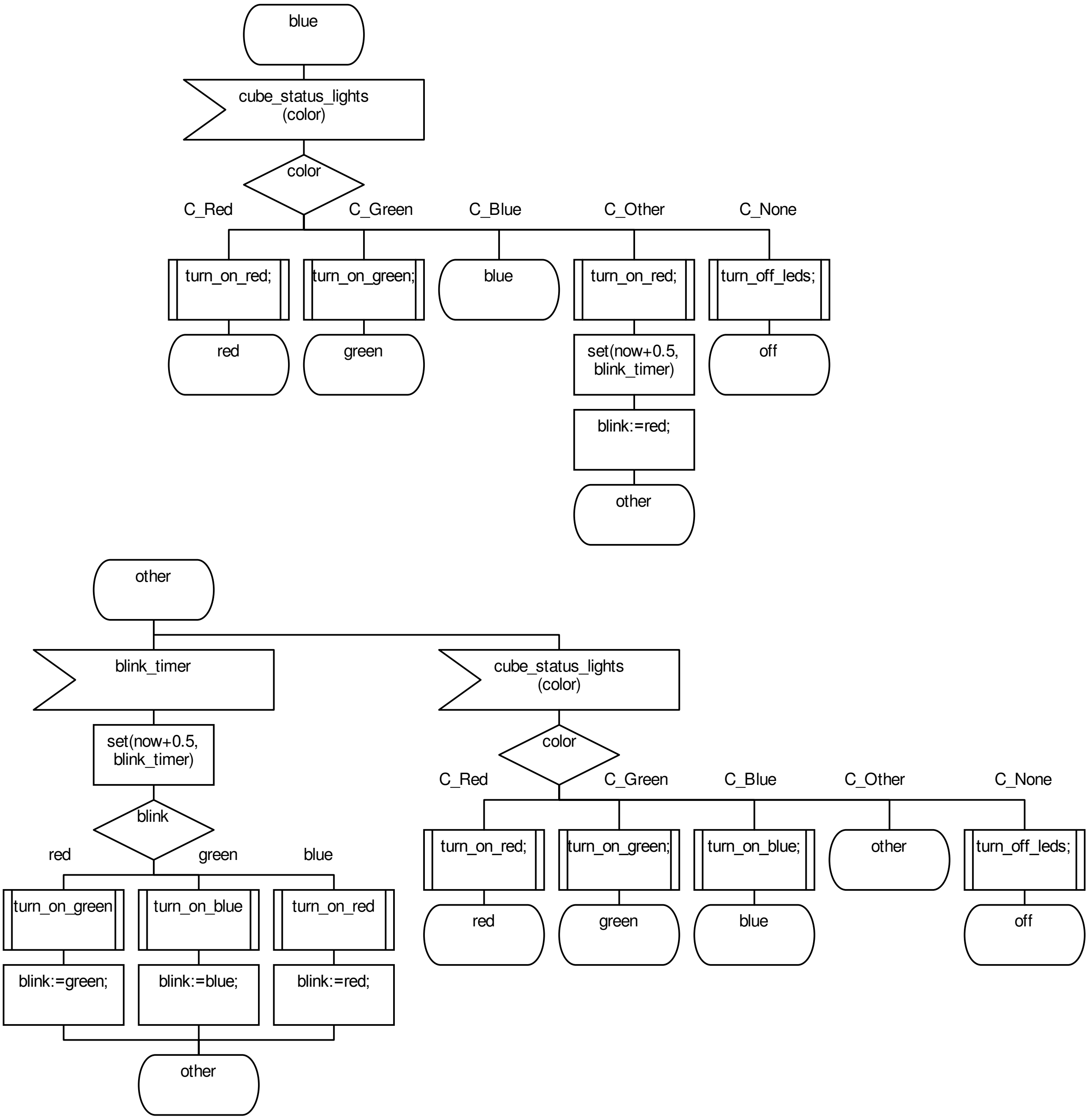

From page 2
Correction _status
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
L_Straight
L_Right
L_Left
L_XtrRight
L_XtrLeft
L_Error
pos_correction (pos_to_correct)
correction_status := pos_to_correct;
line_error
emptySignalQueue (pos_correction)
pos_to_correct := L_Straight; stepper_right := S_Still; stepper_left := S_Still; speed_L := 0; speed_R := 0;
stepper_protocol_left (stepper_left, speed_L, stepstodo)
stepper_protocol_right (stepper_right, speed_R, stepstodo)
start
Correction _status
L_Left
L_Right
L_XtrLeft
L_XtrRight
L_Straight
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
set(now+ TIMERDUR_LONG_1, correctiontimer)
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_SLO... speed_R := SOFT_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := SOFT_CORR_SPEED_FAST; speed_R := SOFT_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_SLO... speed_R := HARD_CORR_SPEED_FAST;
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := HARD_CORR_SPEED_FAST; speed_R := HARD_CORR_SPEED_SLO...
stepper_right := S_Forward; stepper_left := S_Forward; speed_L := NORM_SPEED; speed_R := NORM_SPEED;
stepper_protocol_left (stepper_left, speed_L, stepst...
stepper_protocol_right (stepper_right, speed_R, step...
last_correction (pos_to_correct)
corr_order := one;
start

**Procedure** emptySignalQueue 1(1)

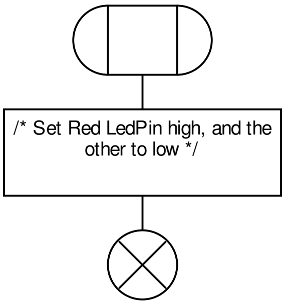



Process statusleds
1(2)
turn_on_red
turn_on_green
turn_on_blue
turn_off_leds
initialize
newType statesType
literals red,blue,green
endnewtype blinkType;
dcl color ColorType;
dcl blink blinkType;
timer blink_timer;
initialize;
turn_off_leds;
off
cube_status_lights
(color)
color
C_Red
C_Green
C_Blue
C_Other
C_None
turn_on_red;
turn_on_green;
turn_on_blue;
turn_on_red;
off
red
green
blue
set(now+0.5,
blink_timer)
blink:=red;
other
red
cube_status_lights
(color)
color
C_Red
C_Green
C_Blue
C_Other
C_None
red
turn_on_green;
turn_on_blue;
turn_on_red;
turn_off_leds;
green
blue
set(now+0.5,
blink_timer)
off
blink:=red;
other
green
cube_status_lights
(color)
color
C_Red
C_Green
C_Blue
C_Other
C_None
turn_on_red;
green
turn_on_blue;
turn_on_red;
turn_off_leds;
red
blue
set(now+0.5,
blink_timer)
off
blink:=red;
other

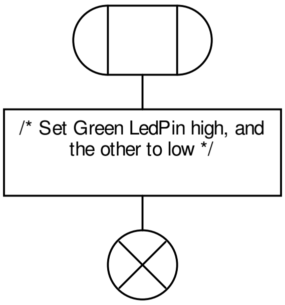
**Process** statusleds 2(2)



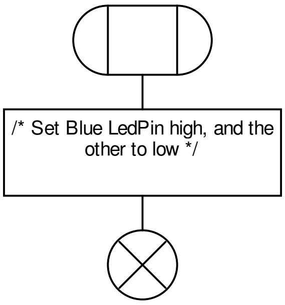
**Procedure** turn_on_red 1(1)



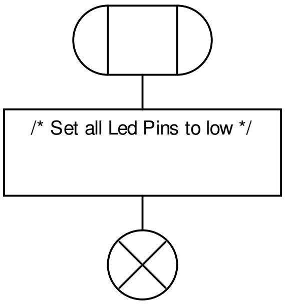
**Procedure** turn_on_green



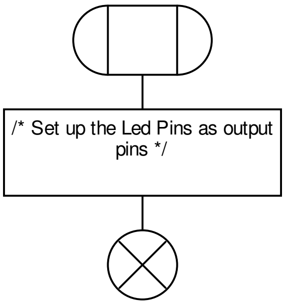
**Procedure** turn_on_blue



**Procedure** turn_off_leds 1(1)

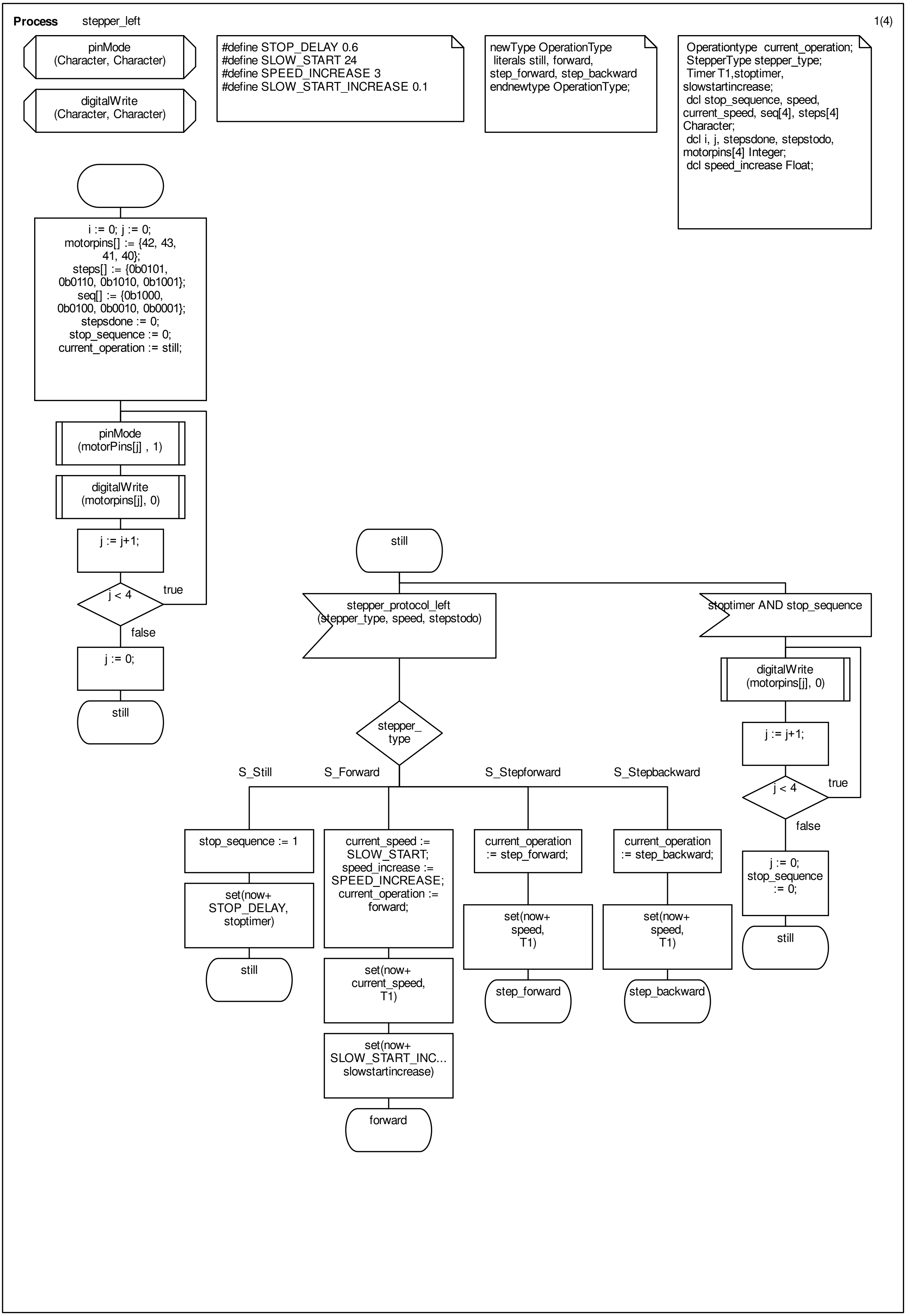


**Procedure** initialize 1(1)

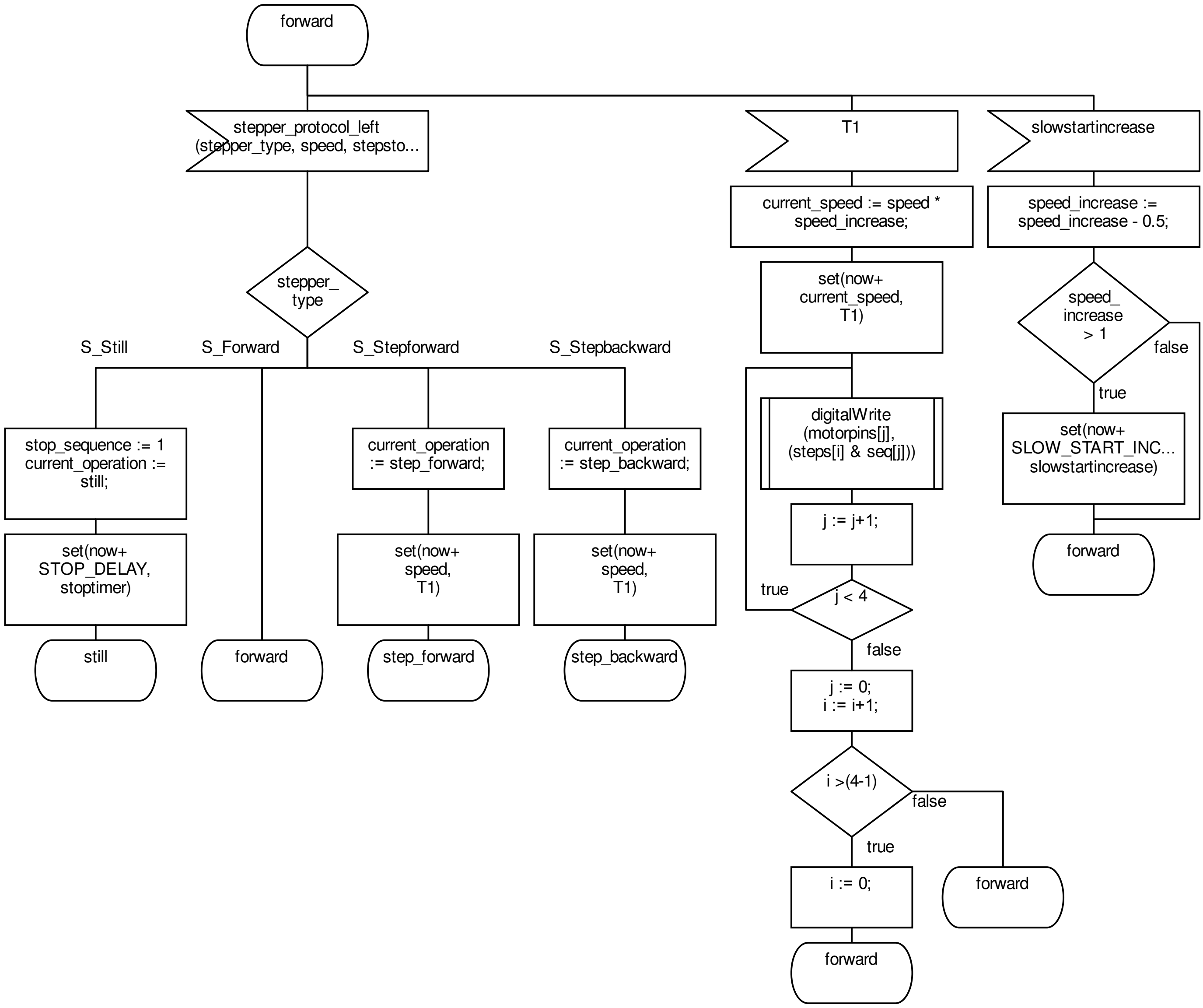


**Process** stepper_right 1(1)

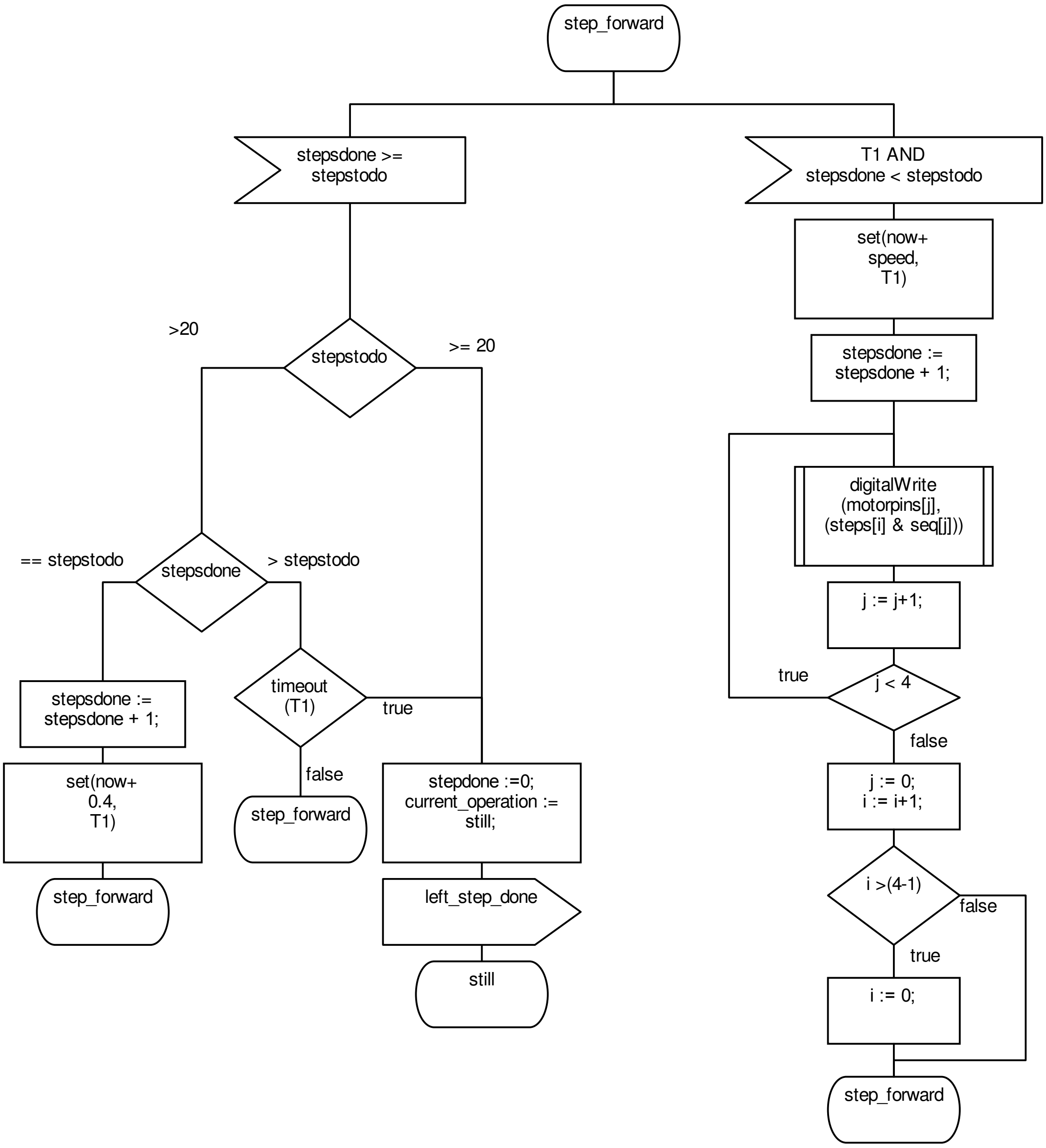
```
/* Similar to stepper_left in all aspects, the only three chages are:
 pin numbers and signals
 motorpins[] = {32, 33, 35, 34}
 sendSignal(left_step_done)
 receivedSignal(stepper_protocol_left,  &stepper_type, &speed, &stepstodo)
*/
```


Process stepper_left
1(4)
pinMode (Character, Character)
digitalWrite (Character, Character)
#define STOP_DELAY 0.6
#define SLOW_START 24
#define SPEED_INCREASE 3
#define SLOW_START_INCREASE 0.1
newType OperationType
literals still, forward,
step_forward, step_backward
endnewtype OperationType;
Operationtype current_operation;
StepperType stepper_type;
Timer T1,stoptimer,
slowstartincrease;
dcl stop_sequence, speed,
current_speed, seq[4], steps[4]
Character;
dcl i, j, stepsdone, stepstodo,
motorpins[4] Integer;
dcl speed_increase Float;
i := 0; j := 0;
motorpins[] := {42, 43,
41, 40};
steps[] := {0b0101,
0b0110, 0b1010, 0b1001};
seq[] := {0b1000,
0b0100, 0b0010, 0b0001};
stepsdone := 0;
stop_sequence := 0;
current_operation := still;
pinMode (motorPins[j] , 1)
digitalWrite (motorpins[j], 0)
j := j+1;
j < 4
true
false
j := 0;
still
still
stepper_protocol_left (stepper_type, speed, stepstodo)
stoptimer AND stop_sequence
stepper_ type
S_Still
S_Forward
S_Stepforward
S_Stepbackward
stop_sequence := 1
set(now+ STOP_DELAY, stoptimer)
still
current_speed := SLOW_START;
speed_increase := SPEED_INCREASE;
current_operation := forward;
set(now+ current_speed, T1)
set(now+ SLOW_START_INC... slowstartincrease)
forward
current_operation := step_forward;
set(now+ speed, T1)
step_forward
current_operation := step_backward;
set(now+ speed, T1)
step_backward
digitalWrite (motorpins[j], 0)
j := j+1;
j < 4
true
false
j := 0;
stop_sequence := 0;
still

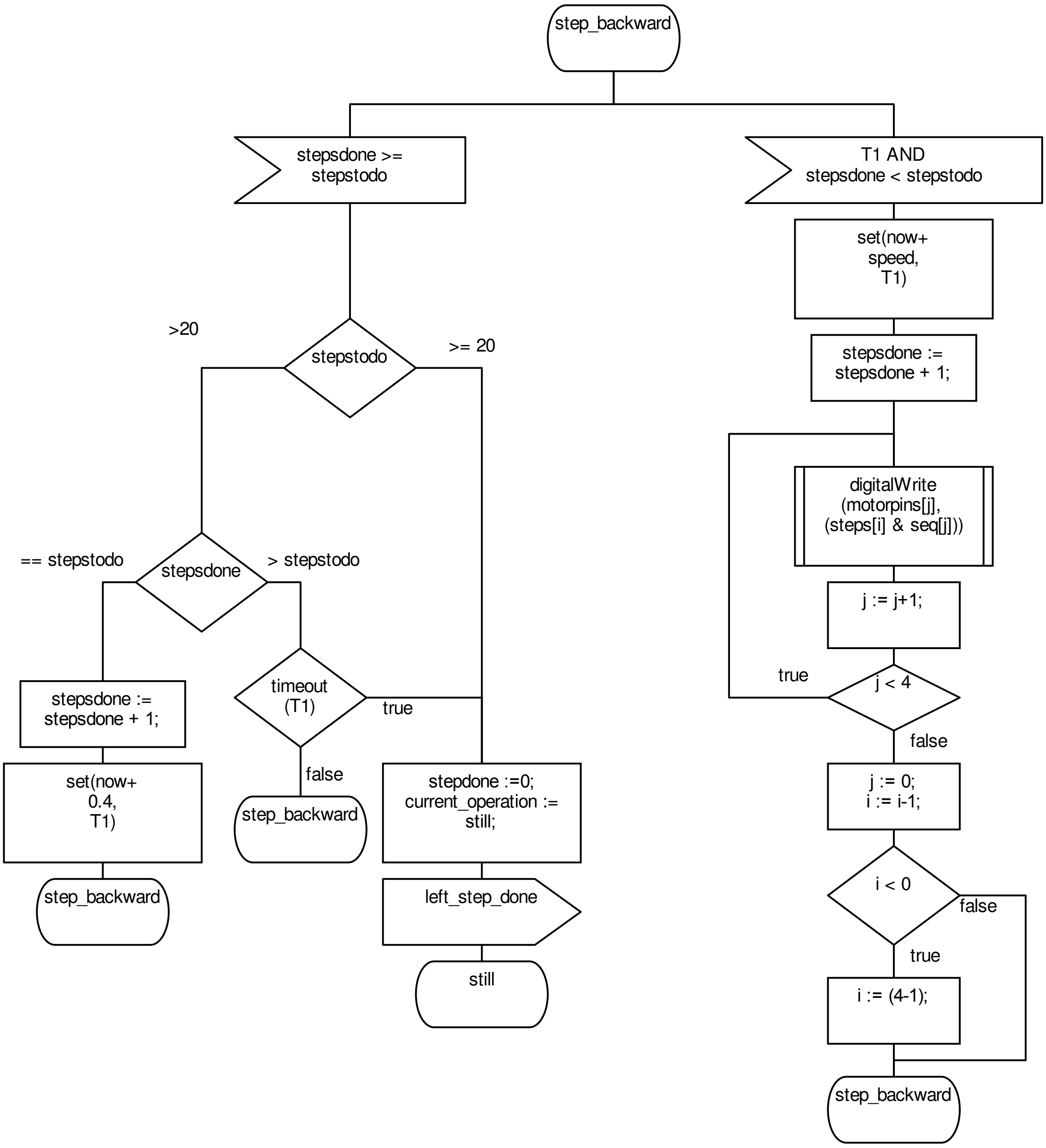
**Process** stepper_left


forward
stepper_protocol_left (stepper_type, speed, stepsto...
T1
slowstartincrease
current_speed := speed * speed_increase;
speed_increase := speed_increase - 0.5;
set(now+ current_speed, T1)
speed_ increase > 1
false
true
set(now+ SLOW_START_INC... slowstartincrease)
forward
stepper_ type
S_Still
S_Forward
S_Stepforward
S_Stepbackward
stop_sequence := 1 current_operation := still;
current_operation := step_forward;
current_operation := step_backward;
set(now+ STOP_DELAY, stoptimer)
set(now+ speed, T1)
set(now+ speed, T1)
still
forward
step_forward
step_backward
digitalWrite (motorpins[j], (steps[i] & seq[j]))
j := j+1;
true
j < 4
false
j := 0; i := i+1;
i >(4-1)
false
true
i := 0;
forward
forward

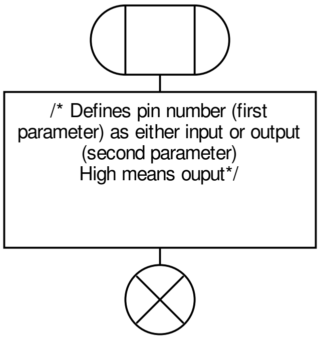
**Process** stepper_left



**Process** stepper_left 4(4)

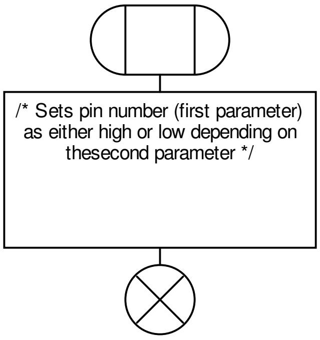


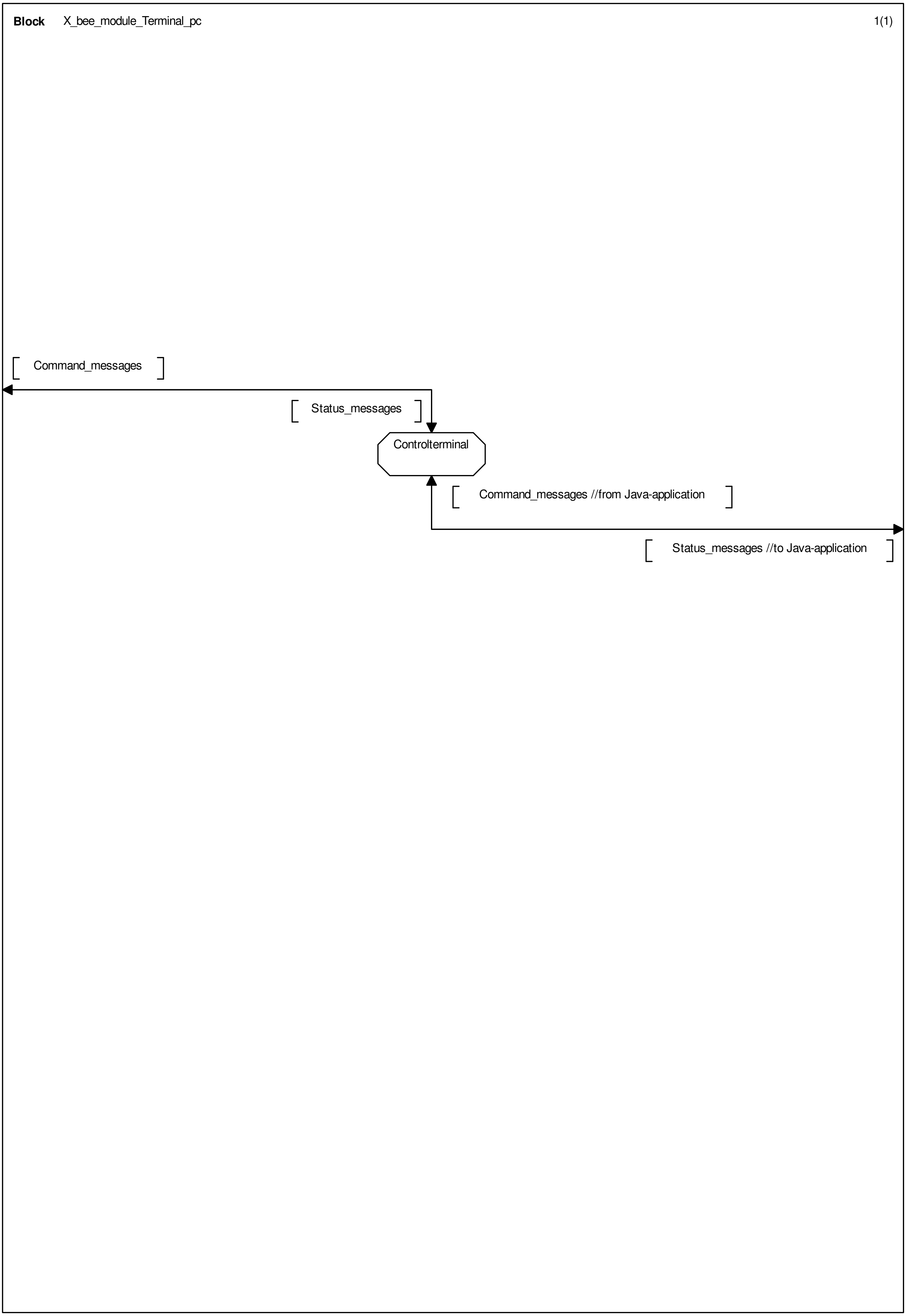
**Procedure** pinMode 1(1)

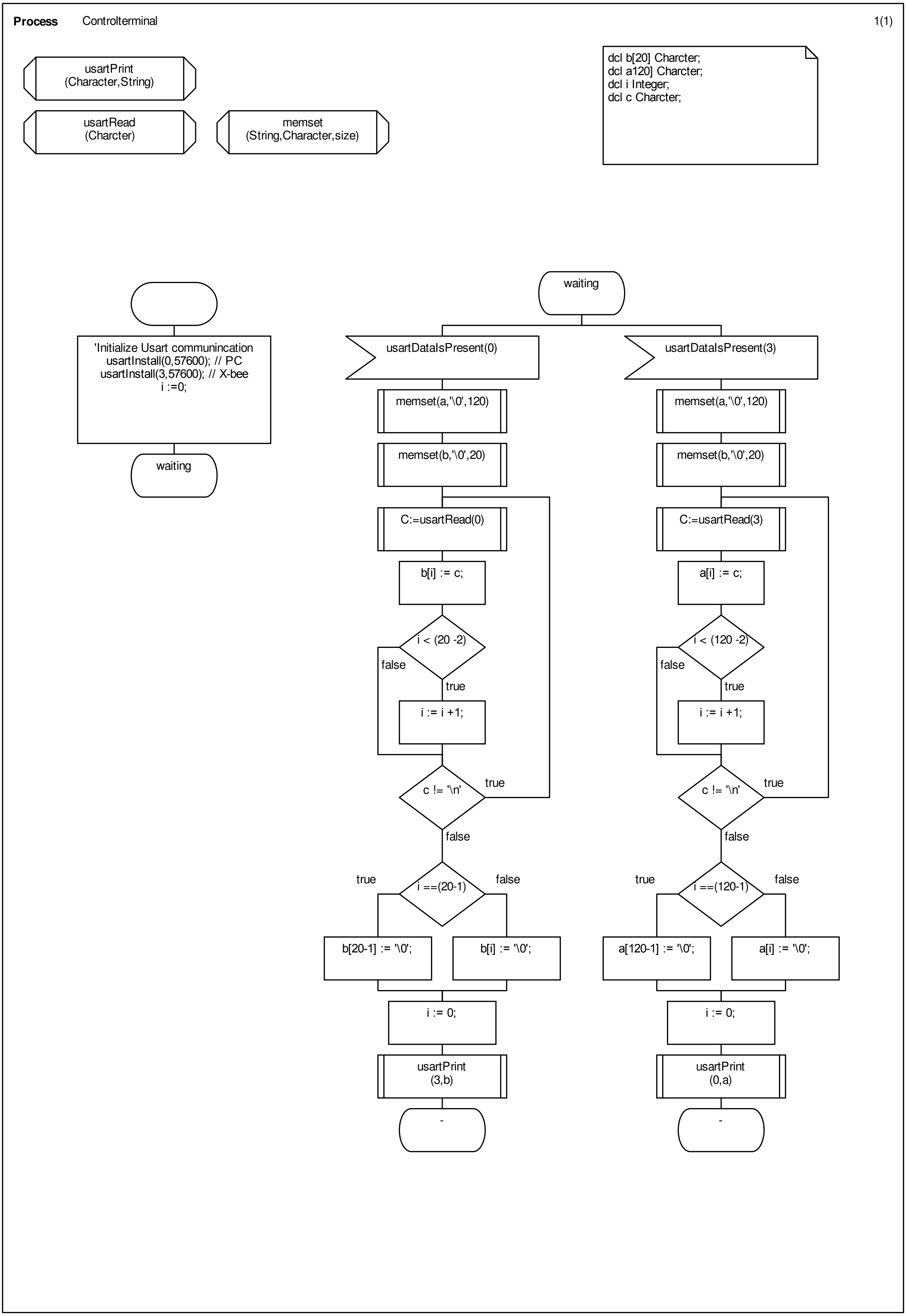


/* Defines pin number (first parameter) as either input or output (second parameter)
High means ouput*/

**Procedure** digitalWrite 1(1)


/* Sets pin number (first parameter) as either high or low depending on thesecond parameter */

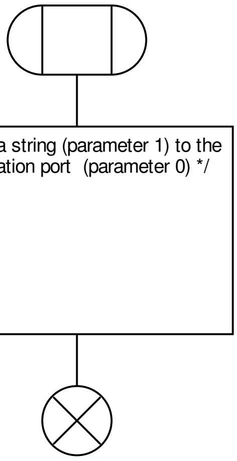

Block X_bee_module_Terminal_pc
1(1)
Command_messages
Status_messages
Controlterminal
Command_messages //from Java-application
Status_messages //to Java-application



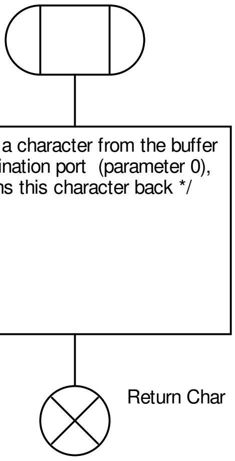
```
dcl b[20] Charcter;
dcl a120] Charcter;
dcl i Integer;
dcl c Charcter;
```

```
'Initialize Usart communincation
usartInstall(0,57600); // PC
usartInstall(3,57600); // X-bee
i :=0;
```

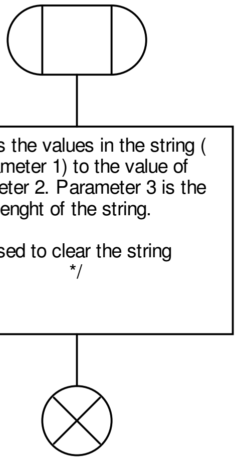
**Procedure** usartPrint 1(1)


/*Write a string (parameter 1) to the Destination port  (parameter 0) */

**Procedure** usartRead 1(1)


/*Reads a character from the buffer of Destination port (parameter 0), returns this character back */
Return Char

**Procedure** memset 1(1)



/* Sets the values in the string ( parameter 1) to the value of parameter 2. Parameter 3 is the lenght of the string.

Used to clear the string */