MyOpenLab Build 3.0.8.4

DIAGRAMS DE F LUJO (quick guide)

For version 3.0.8.3 and above

Prof. José Manuel Ruiz Gutiérrez
j.m.r.gutierrez@gmail.com

Index

1. Justification of this bookstore.

The incorporation of this library in the MyOpenLab tool gives it an important value since it allows you to design simulations oriented to process control through a language widely used in real programming systems of PLC's, process control systems, microcontrollers (PIC's), etc...

Most control processes can be designed using functional blocks arranged in the form of a "flow chart" thus making it much easier to understand how they work and to detect errors.

MyOpenLab incorporates this library fully compatible with the rest of the elements of the other libraries and with the purpose, perhaps in the future, of being able to associate the applications made with its functional blocks to data acquisition cards connected to the PC port, in the same way that other applications similar to this one do.

The didactic possibilities that are added with this library are very important, since the user will hardly need to have any knowledge of electronics or electricity to be able to perform successful simulations.

Very important note:

Important syntax modifications for the new Flowchart library

Formerly	**Now**
=	==
Value type dbl 10	Value dbl rate 10.00
sin(x)/cos(x)/...	Math.sin(x)/Math.cos(x)/.. ..
AND & operator	AND operator &&&
Operator OR |	Operator OR ||

2. How to use the components of the FlowChart library.

The FlowChart library consists of the components shown in Figure 1.

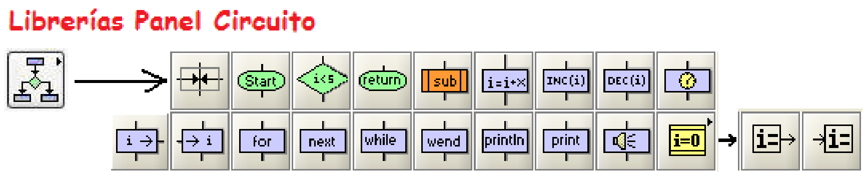


Figure 1

In the following table we place the drawing and the name of each of the blocks.

Union Knot	Decision	Evaluate Expression	Increase	Decrease
Delay	Write	Read	Delivery Data	Collect Data
Home	Return	Routine	Print	

The steps to follow to create an application with this library are as follows:

2.1. Define the variables to be manipulated in the process.

This step is the first one and consists simply in defining the different variables that we are going to handle. There are three types of variables for this version of the library:

Boolean (Binary)
Double (Decimal)
String (character string)

The definition is carried out by means of the "Define Variables" tool that can be invoked from the menu button bar or from the drop-down menu (Figure 2):
VM-> Define variables

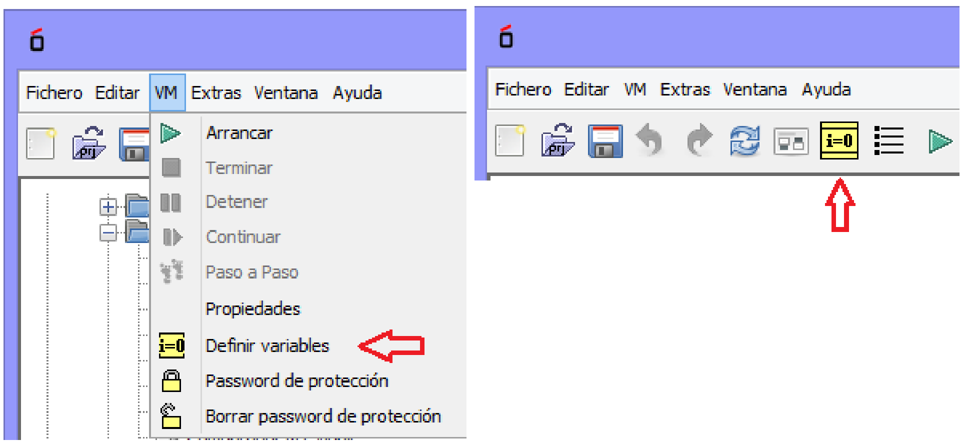
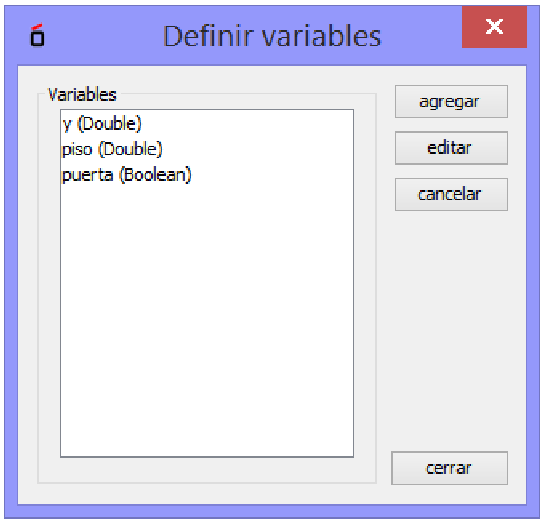


Figure 2

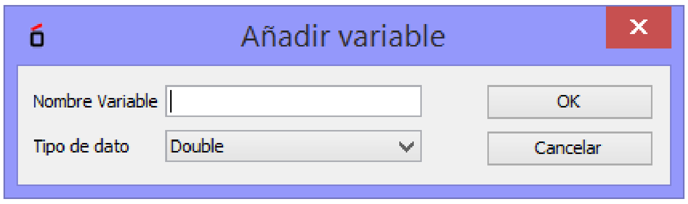
This option, once invoked, opens a window like the one shown in figure 3 through which we can define and/or edit the variables.



y (Double)
piso (Double)
puerta (Boolean)

Figure 3

Using the Add button, we add a new variable through the window shown in Figure 4.



Nombre Variable:
Tipo de dato: Double

Figure 4

Click on the "Data type" menu button to select the data type, as shown in Figure 5.

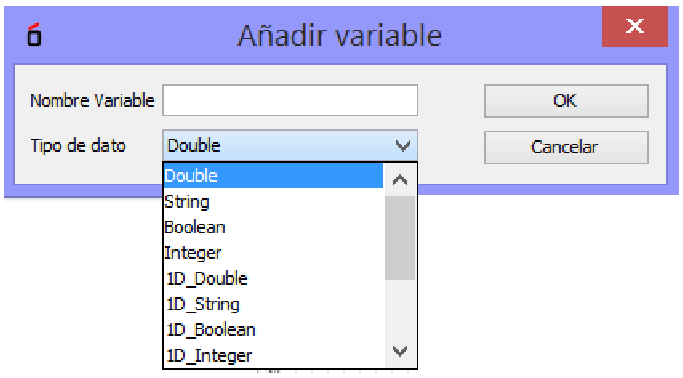


Figure 5

2.2. To make the "flow chart".

Once the variables have been defined, we will proceed to make the "flowchart" by dragging the components from the library area and placing them in the "Circuit Panel". It is recommended to place (figure 6):

- (1) Blocks and the configuration of the values of each one.
- (2) Linking the Blocks together
- (3) Place the external elements (button "run" and box "constant input" - value -i-. Block "Data delivery" and block "Numerical output".

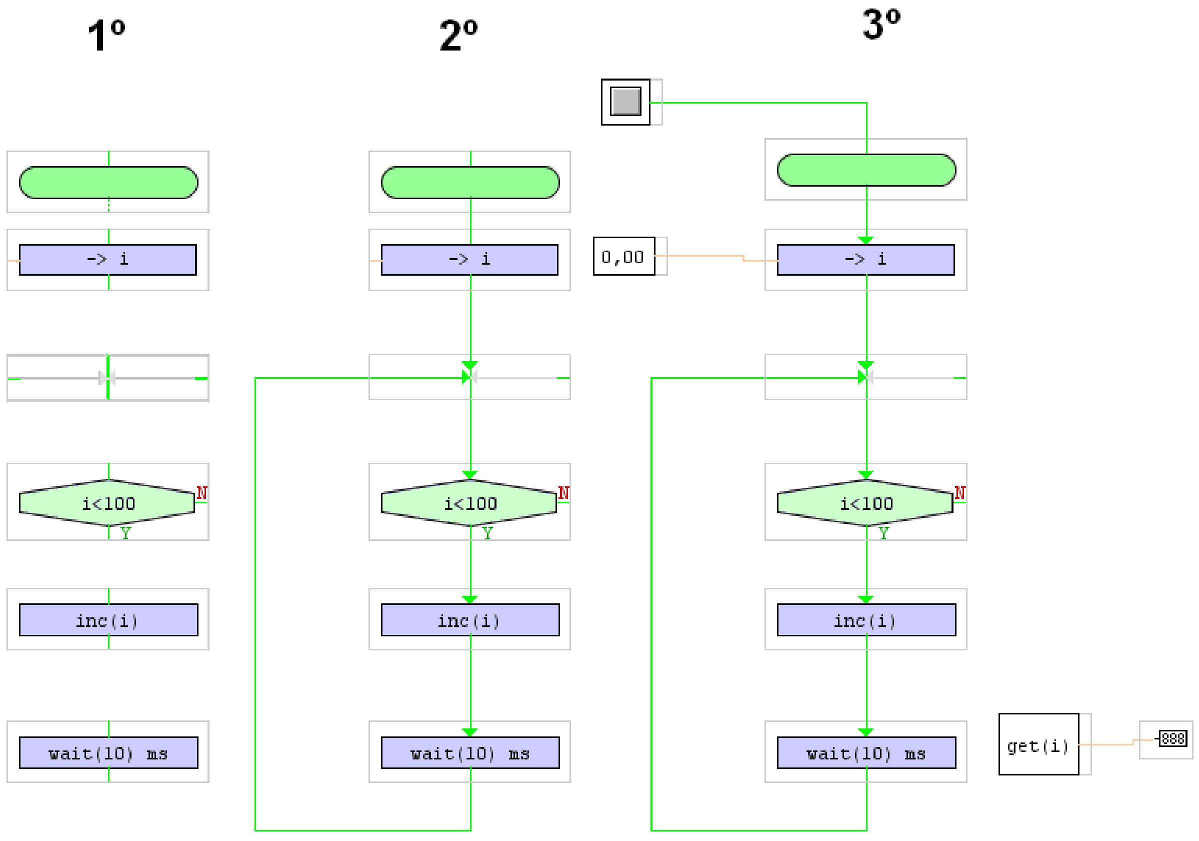
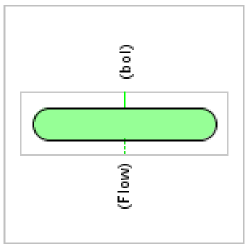


Figure 6

We must take into account that in all Flow Diagrams a Boolean type signal must be placed, which is the one that activates the execution of the flowchart. In the previous figure we can see that we have placed a button for this purpose.
The blocks shown in Figure 6 have the following configuration parameters:

Start"	block is placed at the beginning of all slack diagrams and subroutines.
Read " block	the name of the variable to be read must be assigned (*i*)
Block "**Union Knot**" this block has no parameters.	
Block "**Decision** "	the conditional expression of the block *i>100* must be placed
Block "**Increment**".	block, variable i is assigned as the variable to increment.
Delay " block	block, we assign the delay time that we want to stop the execution of the flowchart once we are simulating its operation

3. **Start" function block**



The Start element is required to start the flowchart and must always be placed at the beginning of the flowchart.

This element represents a Function or Procedure that should always be used also for the start of a subroutine.

This block's input must be activated by a signal of type Boolean (bol) and its output is of type (Flow), which is the data type used for flowcharts.

It is very important to take this detail into account since the connection of the diagram with the rest of the objects in our model must be made according to this concept.

Figure 7

In case this element is part of the beginning of a Subroutine or Procedure, a series of data can be defined as an input variable (in order to collect values from the other parts of the flowchart).

Let's take an example. Suppose we want to define a subroutine called *Test* that handles variables *a* (double), variable *b* (string) and variable *c* (boolean). In this case we would write as parameters of the Start element the text: *Test(double a, string b, boolean c)*

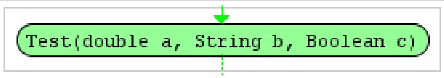
Example :	 Test(double a, String b, Boolean c)

Figure 8

Test is the name of the "Procedure/function".

All parameters are "local" type variables that can be used in the flowchart.

You can use the Start element as a "function" that returns a result. By simply placing

at the end a "Return" function

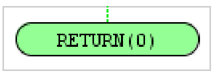

RETURN(0)

Figure 9

3.1. Here is a simple example with the Start function.

This involves printing some data on the terminal using the function *"println"*.

The program is executed by pressing the *OK* button .

The first order that the system receives is to execute the *Test* subroutine, this is done with the "Procedure" function in which we define

y=Test(5,3)

In this case *y* will be the variable in which the value returned by the execution of the Test Procedure will be collected.

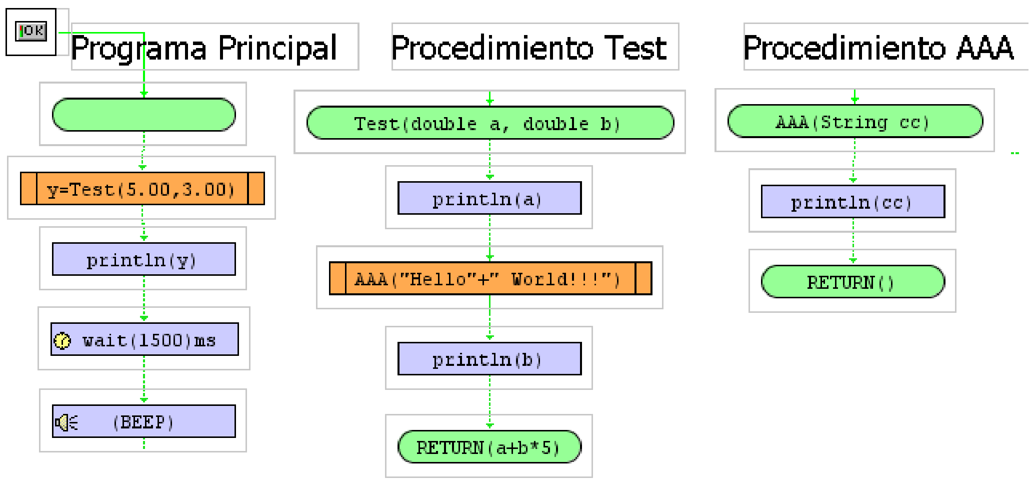


Figure 10

The main program, through the "Procedure" Test function, delivers the following values

a=5 and b=3 to the procedure Test

The "Start" function, with which this procedure must necessarily begin, is shown with the text

Test(double a, double b)

where it is specified which variables are collected (a and b).

The procedure ends with the instruction *"RETURN(a+b*5)"* which returns the value to the instruction that invoked it *y=Test(5,3)* .

Test procedure execution

Once the Test procedure has been started, the first thing that is done is to print the value of the variable to *"println a"* which is carried out in the Output Terminal (see figure ...)

The following operations are a call to a second procedure or subroutine:

AAA("Hello "+"world !!")

In this case, this instruction of type *"Procedure"* calls the *AAA* routine and at the same time passes a string parameter *(Hello world !!!).*

E xecution of the AAA procedure

The *AAA* procedure starts with its *"Start*" instruction by collecting the string sent to it and associating it to the *cc* variable.

"AA(string cc)"

The next instruction will be to print on the terminal the value of cc and this is done by means of:

"println(cc)"

This operation ends the AAA procedure with RETURN, returning the execution control to the instruction

"println(b)"

belonging to the Test procedure

The next instruction to be executed is

*RETURN(a+b*5)*

Once the Test procedure has been executed, control of the program passes to the main program.

The value *y* is then printed on the terminal.

"println(y)"

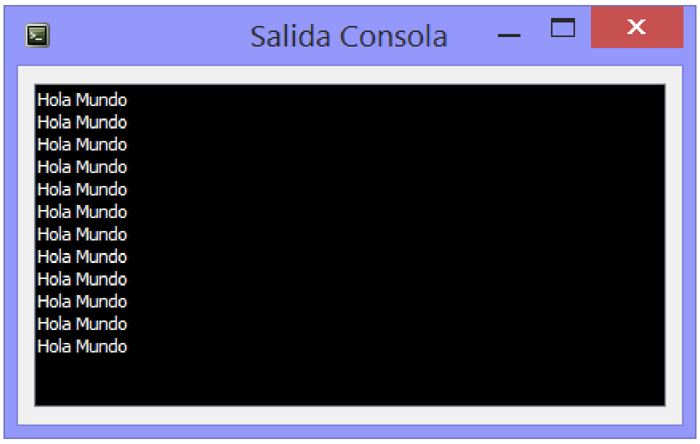
This is followed by a 1500 ms wait. By means of the instruction

"*wait (1500 ms)"*

Finally, the BEEP instruction is executed and the program ends.

"BEEP"

The figure shows the appearance of the terminal window once the program is finished.



```
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
```

Figure 11

To study the program execution flow more closely, you can use the trace option (Debug Mode).

4. Function block "Union Knot".

This block is used to connect up to four lines within a flow diagram. All signals that flow into it are of flow type.

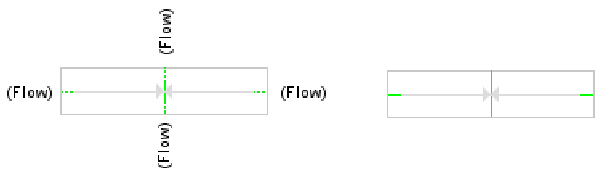


Figure 12.

5. Decision" function block: S y n t a x

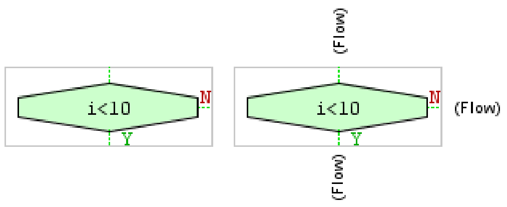


Figure 13

This is a conditional block that activates one of its two outputs (YES "Y" or NO "N") depending on whether an established condition is met.

The condition can be assigned according to the following syntax.

<	**minor**
<=	**less than or equal to**
>	**more**
>=	**greater than or equal to**
==	**same**
<>	**different**

Figure 14 shows a typical application where a conditional block appears asking for the value of the variable "I" and its value is displayed while it is incremented and timed as long as this value is less than 10 "i<10" when it ceases to be, the display-increment-timing sequence will stop executing.

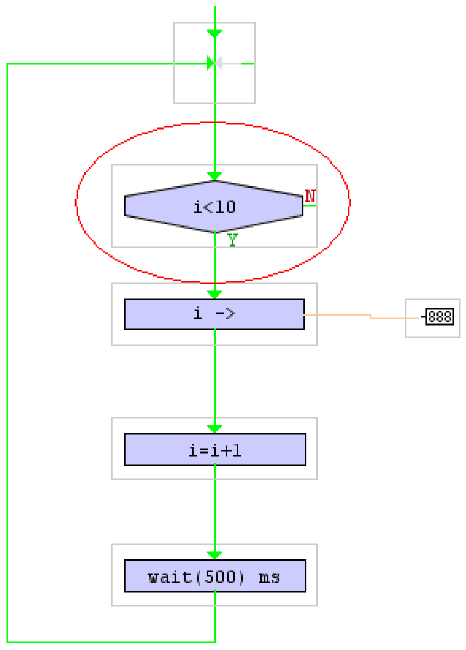
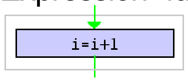
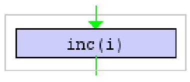


Figure 14

It is important to note that in this case the "Increment" function has not been used for incrementing, but simply with the "Evaluate Expression" function, i.e.

we could replace the block  with the block .

6. Function Block "Evaluate Expression".

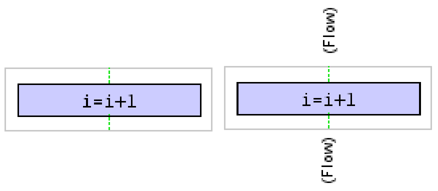


Figure 15

This block can be used to evaluate mathematical expressions containing previously defined variables subject to the operators listed below. This component supports **Boolean, Double and String** variables.

Care should be taken in the preparation of an organization chart not to mix different types of variables.

The operators that can be used in this function block are:

Boolean Operations:

& &	**Logical product AND**
||	**OR logic sum**
!	**Negation NOT**

Example:

A=TRUE, B=FALSE

Y=(!A) & & B = ((NOT TRUE) AND FALSE) = FALSE AND FALSE

Result : Y= FALSE

Operations with double numbers "with decimals".

%	**Module**
^	**Power (x^y)**
+	**Sum**
-	**Subtract**
*****	**Multiplication**
/	**Division**

Example :

y = x+z*4-1;

Numerical functions

Math.sin(x)	**Breast**
Math.cos(x)	**Cosine**
Math.tan(x)	**Tangent**
Math.asin(x)	**Seno Arc**
Math.acos(x)	**Cosine Arc**
Math.atan(x)	**Tangent Arc**
Math.asinh(x)	**Hyperbolic Sinus**
Math.cosh(x)	**Hyperbolic Cosine**
Math.tanh(x)	**Hyperbolic Tangent**
Math.abs(x)	**Absolute Value**
Math.log(x)	**Logarithm (decimal)**
Math.ln(x)	**Loga natural rate (neperian)**
Math.exp(x)	**Exponent**
Math.sqrt(x)	**Root**
Math.r ound(x)	**Random**
Math.fak(x)	**Factorial**
Math.toDeg(x)	**Converts radians to degrees**
Math.toRad(x)	**Converts degrees to radians**

All functions carry the particle Math e.g.: **Math.sin(x)**

String operations

use strings with quotation marks : "String".

+	**Sum of chains** **Example : "Hello _" + "world"** **Result= "Hello _world".**

7. Increment" function block

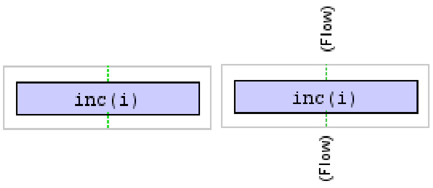


Figure 16

This block allows you to increment a variable specified in it.
The applications with this block can be very diverse and among them we can highlight the counting of events in a process. The variable must be of type "dbl" (Decimal).

8. Function block "Decrement".

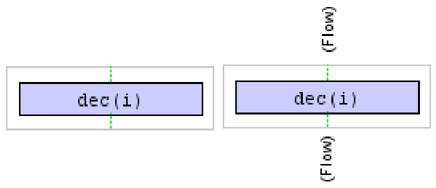


Figure 17

This block does exactly the same as the previous one but in its case it decrements the variable to which it is associated. The variable must be of type "dbl" (Decimal).

9. Delay" function block

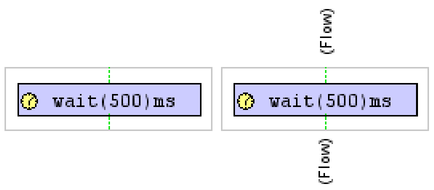
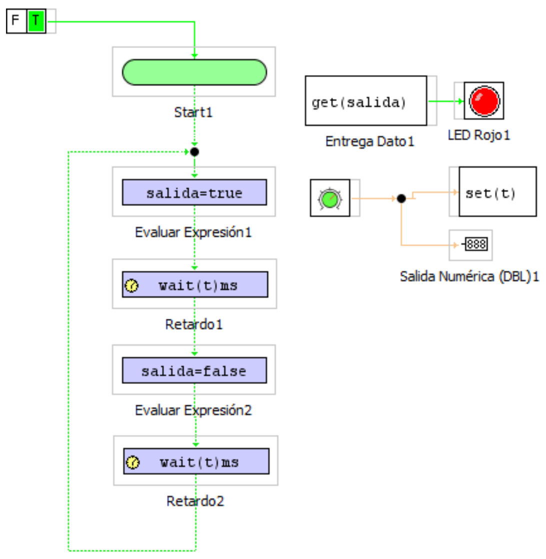


Figure 18

This block introduces a delay in the execution of the program determined by the value in ms of the parameter we enter.

It is used to generate impulse of variable duration, counting, timing, etc.

The time value can be assigned through a variable, as shown in this example:



10. Write" function block

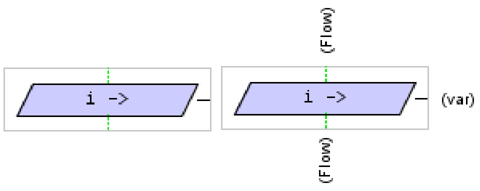


Figure 19

This block allows to deliver a value of a bol, dbl or str. type variable to the system in order to display or process it in other library blocks.

In Figure 20 we see how variable I is deposited in a numerical output box.

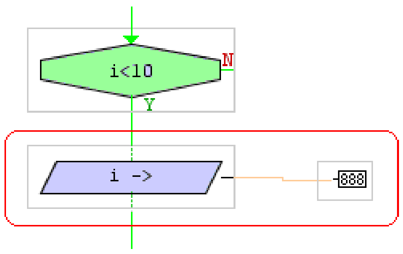


Figure 20

11. Read" function block

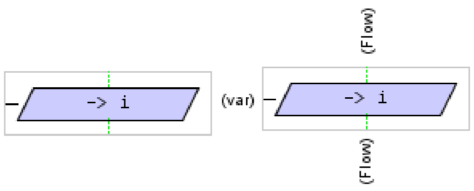


Figure 21

This block allows to collect (read) a value of a bol, dbl or str. type variable to the system in order to be able to make use of it (process it) in the flowchart.
In Figure 22 we see how the variable "j" is taken from a "constant" block.

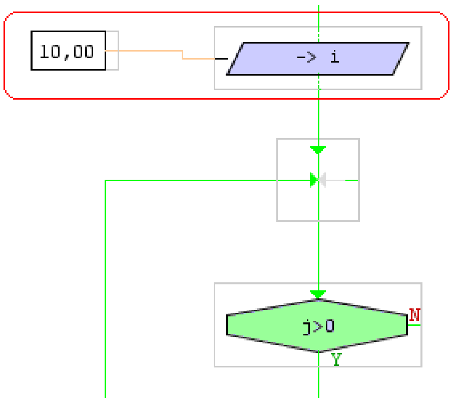
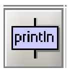


Figure 22

12. Function block "Output (write data)".



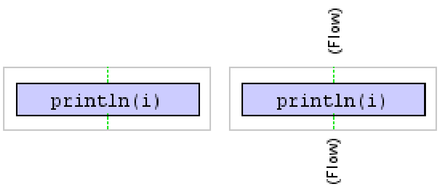
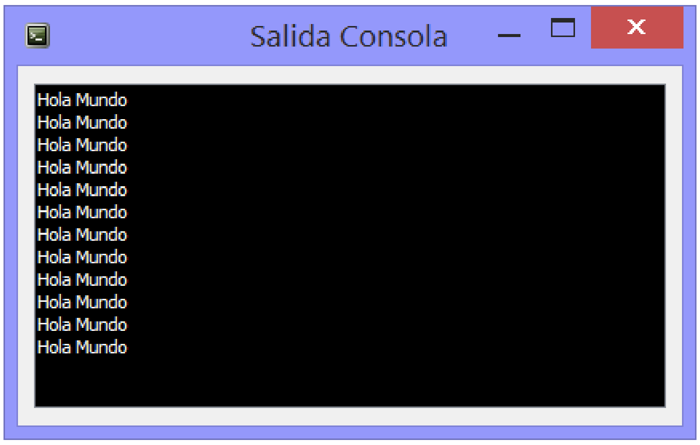


Figure 23

This element prints the value of a variable in the "Output" window.



```
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
Hola Mundo
```

Figure 24

Usage: :

Println(<expression>)

Example : Println("Hello World!"). Println(a) (a = double)

13. Function block "Procedure(routine)".

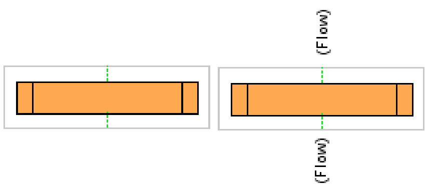


Figure 25

This element will always invoke a subroutine or procedure that will always have a structure starting with a "Start" function and ending with a "RETURN" function.

In this case the Start function does not need to be activated with an activation input (bol type signal) since it will be activated automatically when the implicit name matches the name of the "Procedure" function.

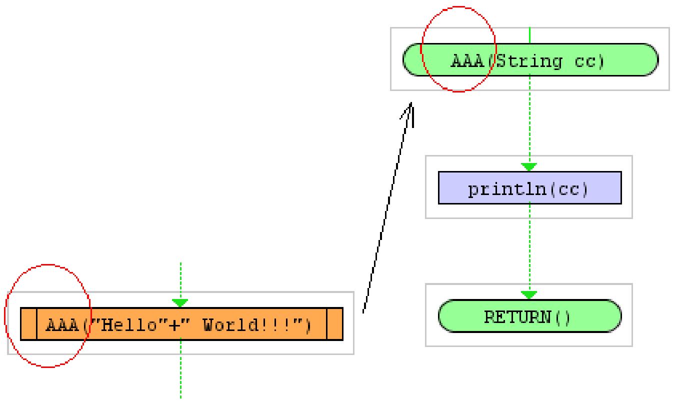


Figure 26

In this case the procedure is called *AAA* and the parameter it passes is a string associated to the variable *cc*. It is seen that when invoking a procedure, values can be given to it, in the same way that the procedure once executed can return values as seen in the following example, in which the procedure passes the values 5 and 3 and collects the value of y that in this case is y=a+b.

The procedure function is of great importance and allows numerous combinations to be made when designing a flowchart.

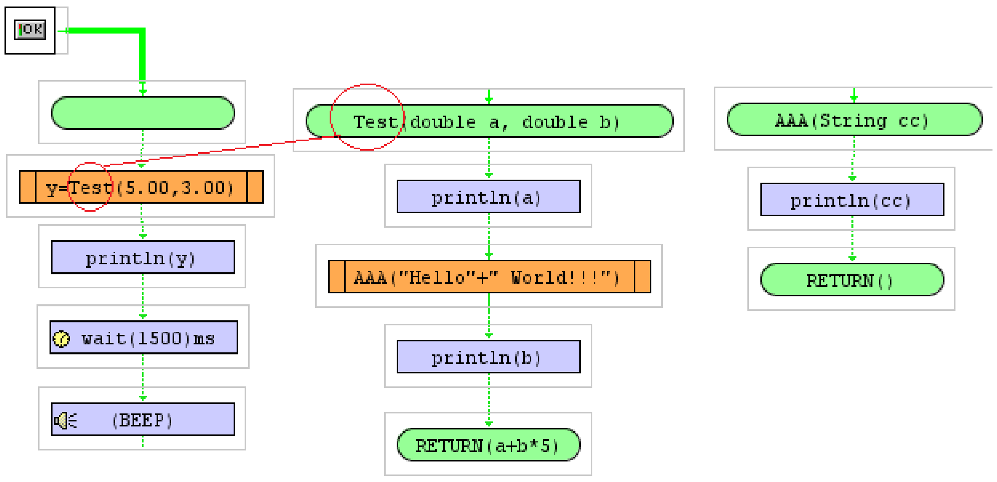
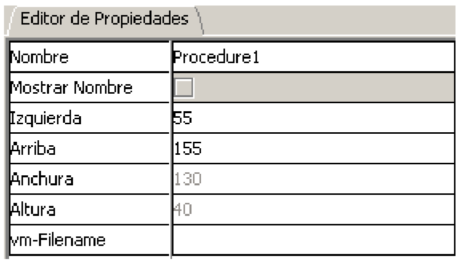


Figure 27

The "Procedure" function can invoke a procedure that is not in the diagram itself, i.e. it can open a function that is in another saved file. To do this, it is sufficient to fill in the vm-Filename field in the Properties editor of the function. The name of the VM to be opened must be entered in this field. The Filename" must contain the name of the VM file (without the "vlogi c" extension).

Example: sub-vm = subvm.vlogic -> the subvm


Editor de Propiedades

Nombre	Procedure1
Mostrar Nombre	☐
Izquierda	55
Arriba	155
Anchura	130
Altura	40
vm-Filename	

Figure 28

14. Function block "Return (Sub. return)".

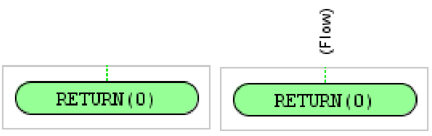


Figure 29

This function is used to close a subroutine or procedure.

You can return a set value

In the diagram we see how the Test procedure returns through the "Return" function the value of a+b*5 which is the value of "y=Test(5,3)".

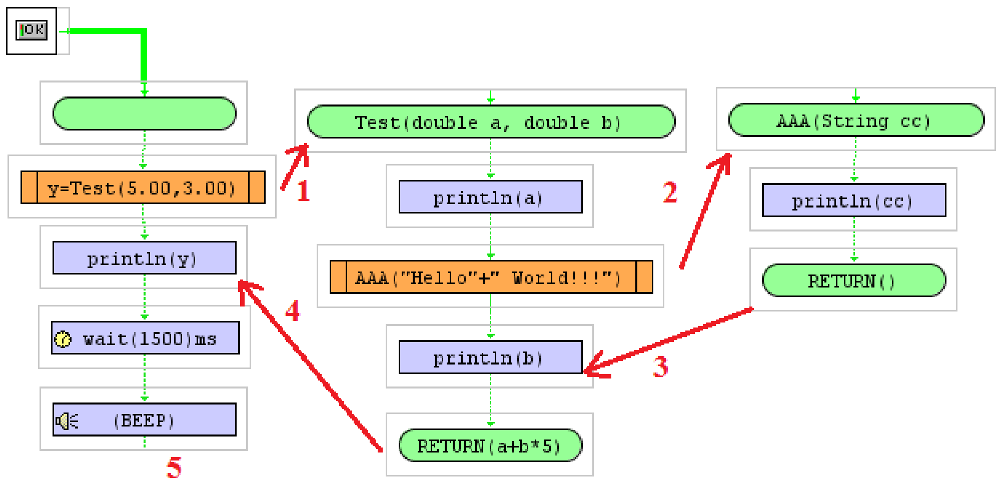


Figure 30

Figure 30 shows the sequence in which the flowchart is executed.

15. Data Delivery" function block

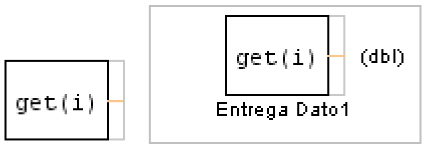


Figure 31

This function is similar to the previous "Write" function since what it does is to send one of the data we have defined to work in the flowchart to any other block of the system.
The difference is that while the "Write" block only reads when the execution sequence passes through the block, here the value is read at all times.

Figure 32 shows a simple example in which this function is used to deliver the value "onx" and output it to a digital output device "led". What the application does is that the Boolean signal "onx" goes from value 1 to 0, staying on for 50 ms and staying off for 500 ms. Emulating a kind of flash.

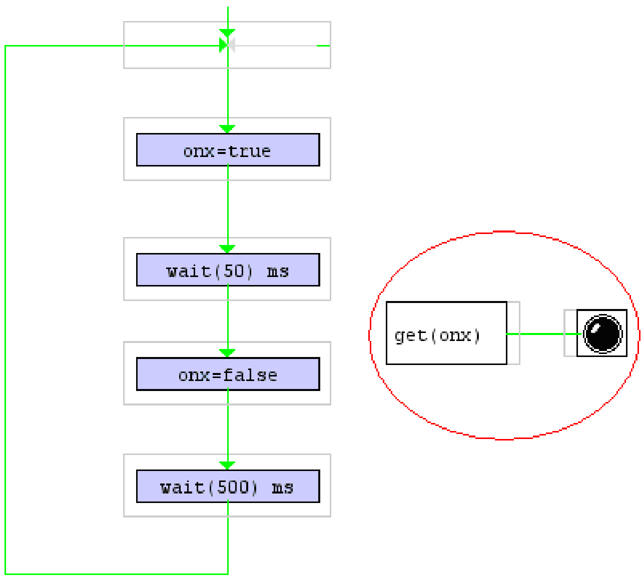


Figure 32

16. Collect Data" function block

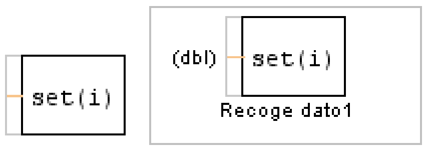


Figure 33

This function is similar to the previous "Read" function since what it does is to collect a data from the system and associate it to a variable of the flowchart. The difference with the "Read" block is that the reading in this block is performed only when the execution sequence passes through the block while in this one it is performed at all times.

In Figure 34 we see an example where the variables "a" and "b" are taken from the control rods a and b and used to be processed by the simple example of a flow chart comparing the value of both

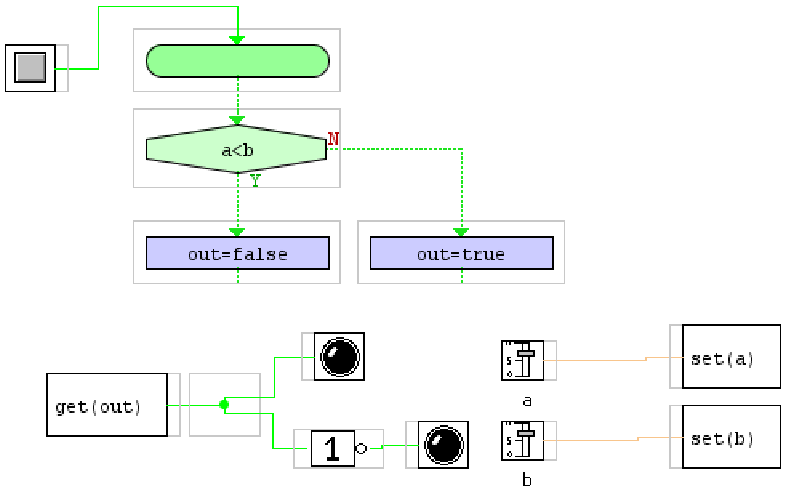


Figure 34

17. Realization of the "tracing" of a flowchart.

MyOpen Lab offers the possibility of tracing the execution of a flowchart, allowing in this case to see how the information evolves throughout the application.

To draw a diagram, you can use the option from the button bar. This option launches the application and runs it slowly. The execution speed can be varied by means of the slider bar shown in figure 35.

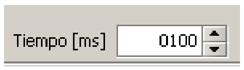

Tiempo [ms] 0100

Figure 35

As the execution is carried out, the function blocks that are activated are marked more intensely, as shown in Figure 36.

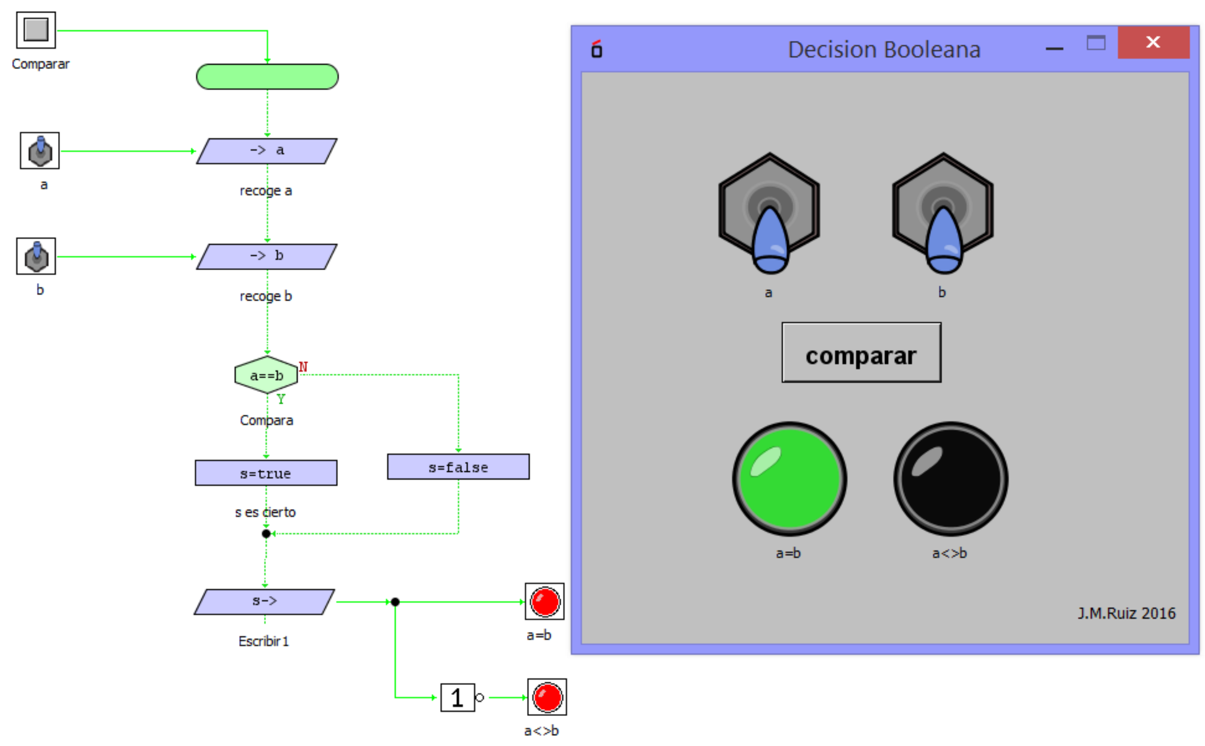


Figure 36

18. Examples of FlowChart library usage

We will now discuss some basic examples in which this library is used. We must not forget that it makes it much easier to make small sequential automatisms and also to exercise the use of algorithms.

18.1. EXEMPL O1

EVALUATION OF VARIABLES OF TYPE BOOL EANO (bool)

Evaluation of two Boolean variables by means of a conditional jump element called **"Decision".** See figure 23

We see that the variables have been defined as **i** and **j** and are collected by two instructions of the type "Read". The decision block evaluates the condition **i=j** and to its outputs we have directly placed two digital value display devices **"Leds".**

The result is that this simple example compares the values and, depending on the result, activates one output or another.

Figure 37 shows the variables in the corresponding window activated by the button in the button bar or the *VM--> Define Variables* option in the menu.

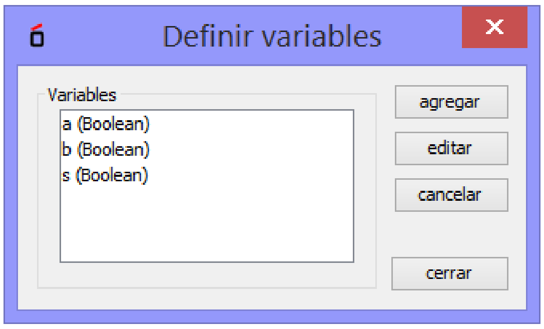


Figure 37

The "start" button must be activated each time we want to perform the comparison function, since once the corresponding "led" is activated, the flowchart is stopped (there is no feedback loop).

Figure 38 shows the window in execution mode.

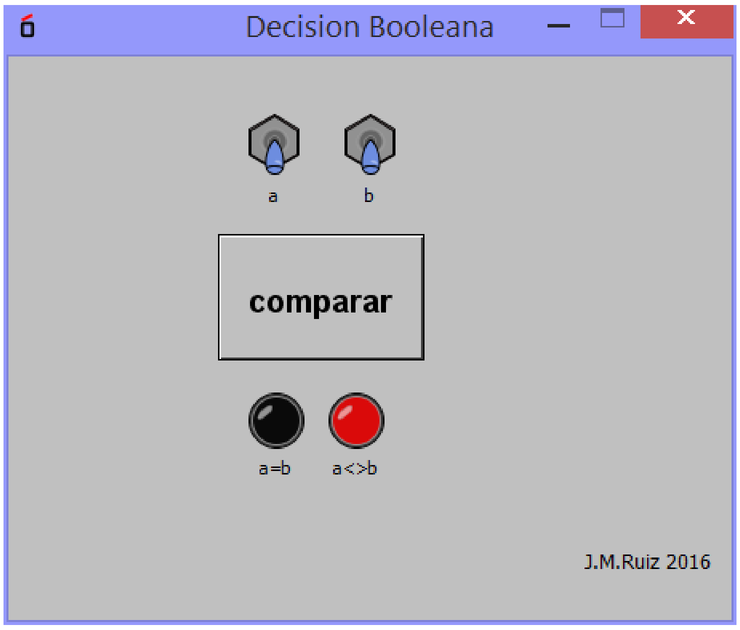


Figure 38

18.2. EXAMPLE 2

LOOPING AND EVALUATION OF DECIMAL VALUES (double)

In this case we perform the same example but instead of comparing Boolean type variables we do it with decimal type variables (double).

In this example, two loops have been placed with two buttons that "fire independently" each of the two flowcharts, which allows us to affirm that two or more flowcharts can be made to coexist and run in parallel.

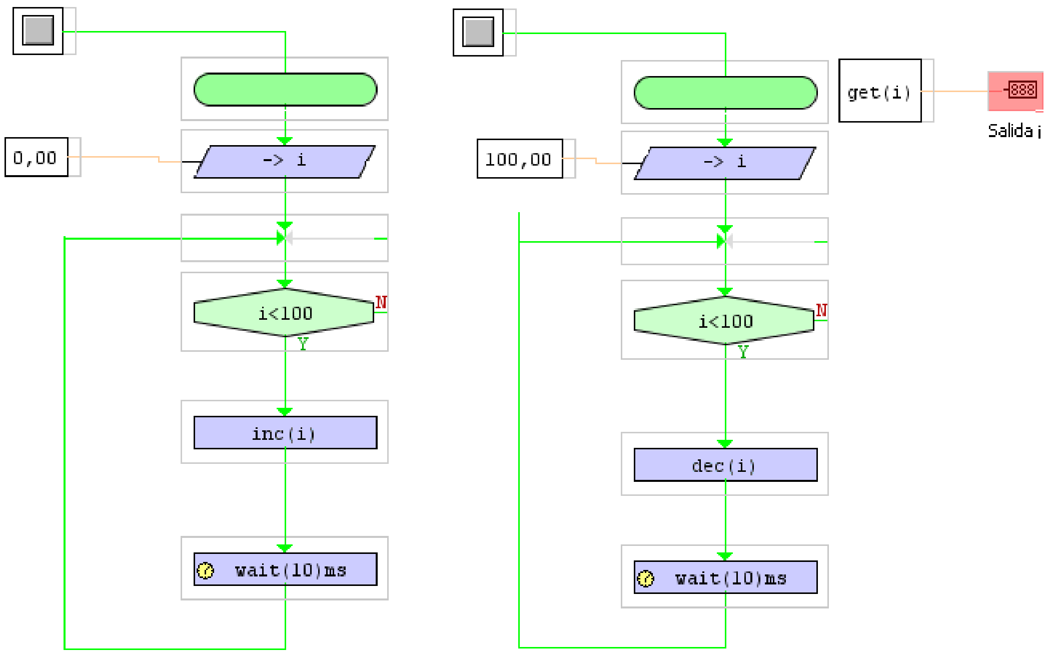
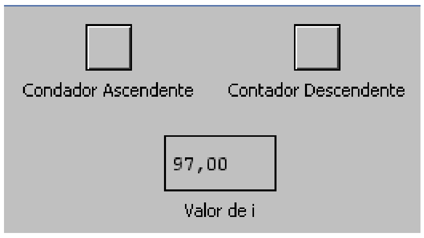


Figure 39



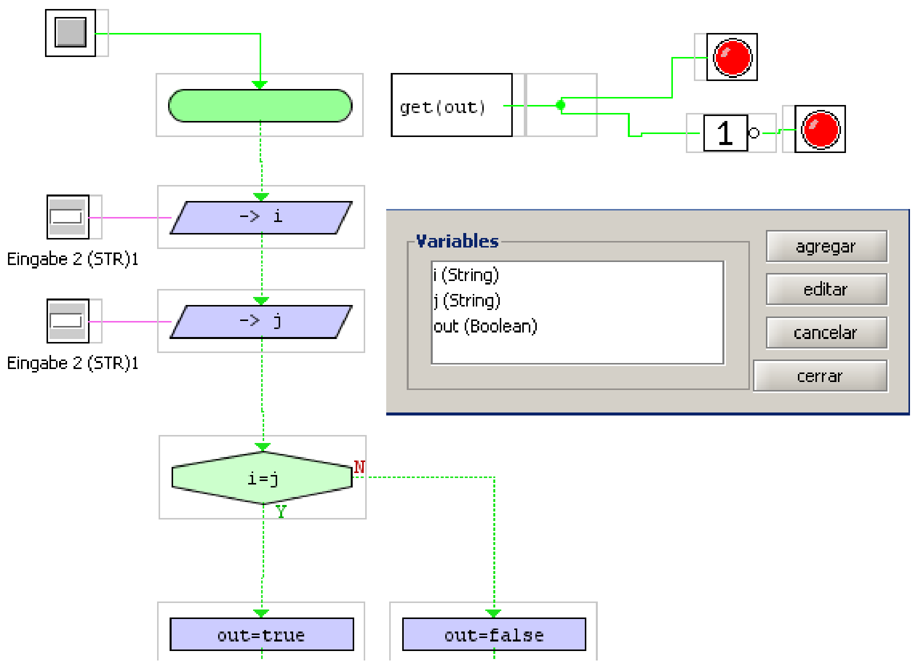
The diagram on the left is an ascending counter from 0 to 100 that is implemented around the count variable **i** that is initialized by the corresponding value 0.00 and is subsequently tested for its value with the condition *i*< 100 by exiting the loop when the set condition is not met. The variable is incremented each time it passes through the function block **"Increment"** *(inc (i))* . A **"Delay"** *(wait (10)ms)* block has been placed in order to be able to visualize how the values are changing.
Figure 40
The flowchart on the right performs the same function but in this case decrementing the value of the variable i., i.e. the count is started by assigning to i the value 100 and decremented until the condition of the **"Decision"** block *i>0* is no longer fulfilled.

Figure 40 shows the display screen

18.3. EXEMPL O3

COMPARISON OF STRING TYPE VARIABLES

In this example we see the case of a comparison of character strings collected through two text collection boxes; Variables **i** and **j**



i (String)
j (String)
out (Boolean)

Figure 41

The result of the comparison is displayed by two LEDs. The comparison is sorted by means of the "compare" button.

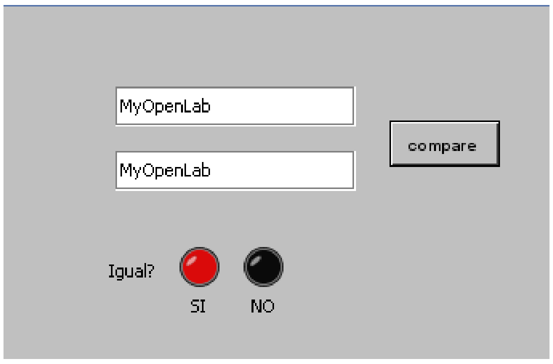


Figure 42

18.4. EXAMPLE 4

CAR GUN LIGHTS

The idea is to create an application that alternately activates two lights that simulate those of an ambulance or police car and also a light that acts in a way that simulates a flash.

For this purpose, two flowcharts are created, having previously defined the variables on and onx as Boolean variables. Figure 43

Variables are collected from the flowchart through the function "Data Delivery" *get(on)* y *get (onx)*

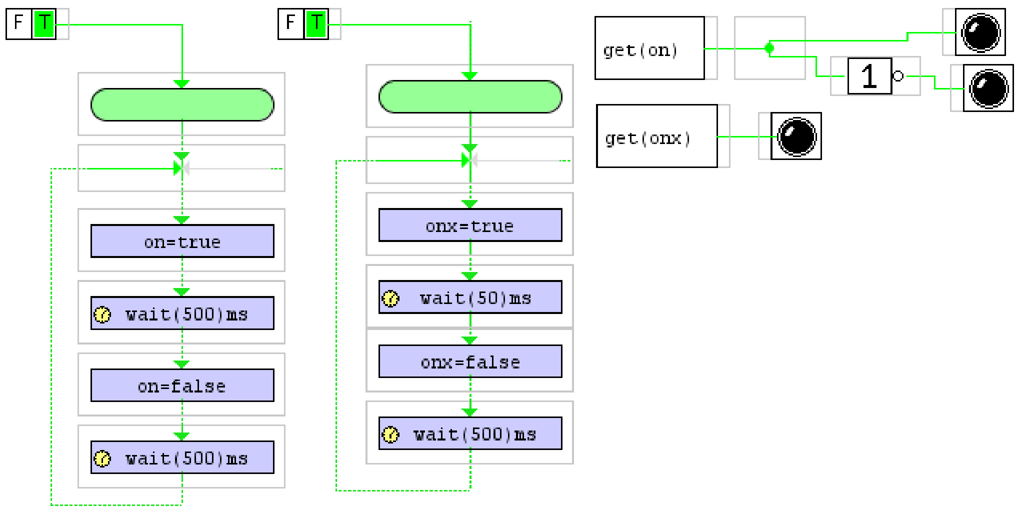
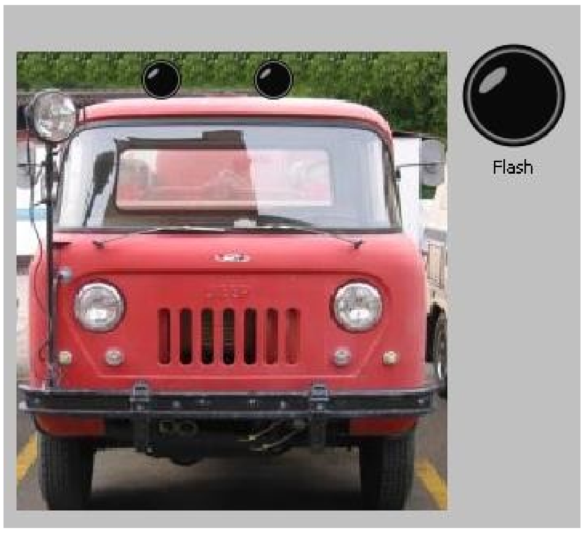


Figure 43

The two flowcharts are of the feedback type (looped) and the "Delay" *wait* instruction is used to achieve the intermittency and flash effects.



In order for the lights to be permanently activated, each flowchart is started with a Boolean constant in each of them.

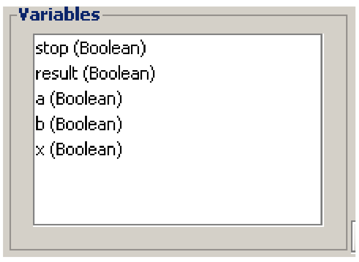
The display screen consists of lamps (LED objects) and an image of a car. Figure 44.

Figure 44

18.5. EXAMPLE 5

DELIVERY AND COLLECTION OF VALUES FROM A FLOW CHART

In this example (Figure 45) we see the possibility of collecting and/or delivering values from a flowchart to the rest of the MyOpenLab functions.



Variables

stop (Boolean)
result (Boolean)
a (Boolean)
b (Boolean)
x (Boolean)

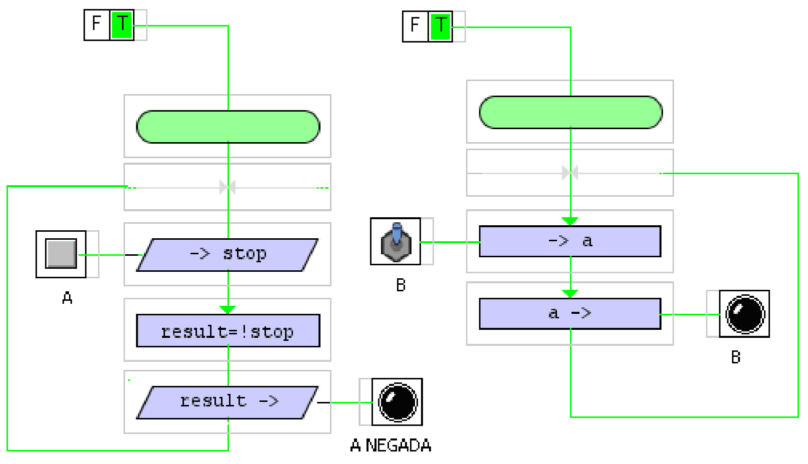


Figure 45

In the flowchart on the left we see how the value emitted by "Button A" is picked up and displayed by the diode "Led A NEGATED".

In the flow chart on the right the variable "B" is collected and delivered to "LED Diode B".

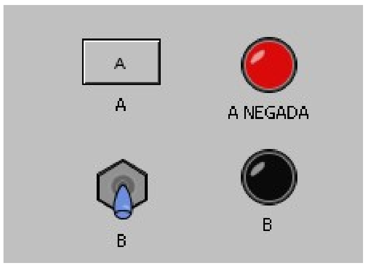


Figure 46

18.6. EXAMPLE 6

SEMAPHORE

In the following example we have simulated the operation of a traffic light in which basically a fixed sequence is established in the switching on and off of each of the traffic light bulbs (red, green, amber).

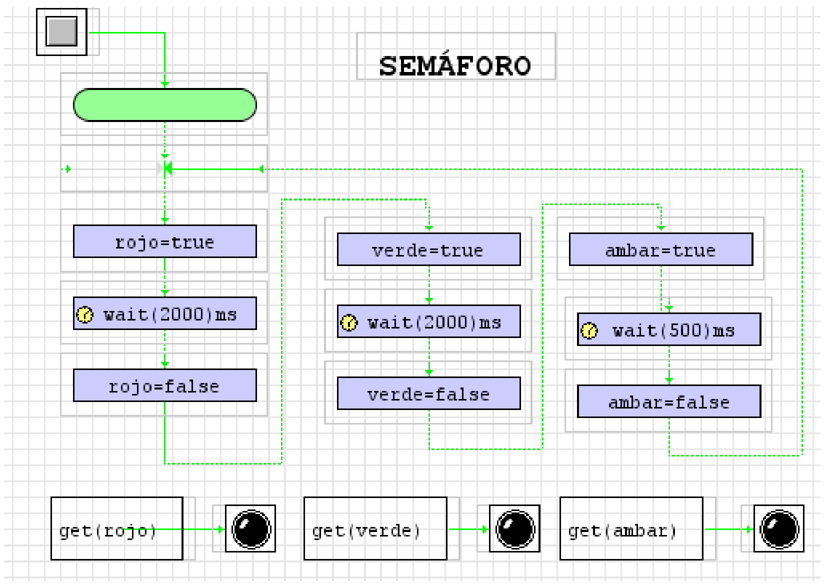
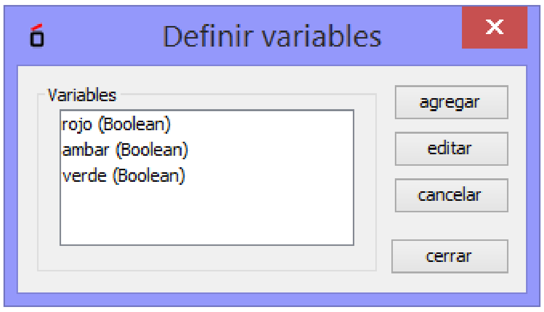


Figure 47

The instructions used are the "Expression Evaluation" instruction in which each of the lamps is set to "true" (on) or "false" (off). On the other hand, the time delay of each lamp is set by means of the "Delay" instruction.

The variables that have been created are shown in Figure 48.



rojo (Boolean)
ambar (Boolean)
verde (Boolean)

Figure 48

The output variables of our system are collected by the function "Data delivery" and the LEDs indicating each of the colors are taken.

Figure 49 shows the visualization panel in the simulation execution state.

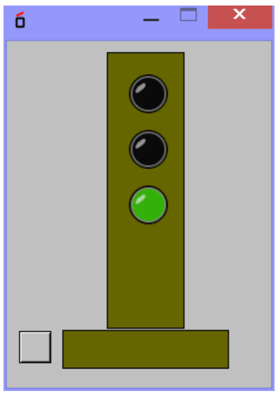


Figure 49

ANOTHER WAY TO PERFORM A TRAFFIC LIGHT SIMULATION

This same simulation model can be carried out using the "library" element.

user". .

This function block incorporates a complete traffic light in which we have as input variables the values of the ignition times of each lamp and its outputs are directly connectable to the lamp indicator LEDs. In figure 50 we see the semaphore block schematic and then the application of this in a simulation:

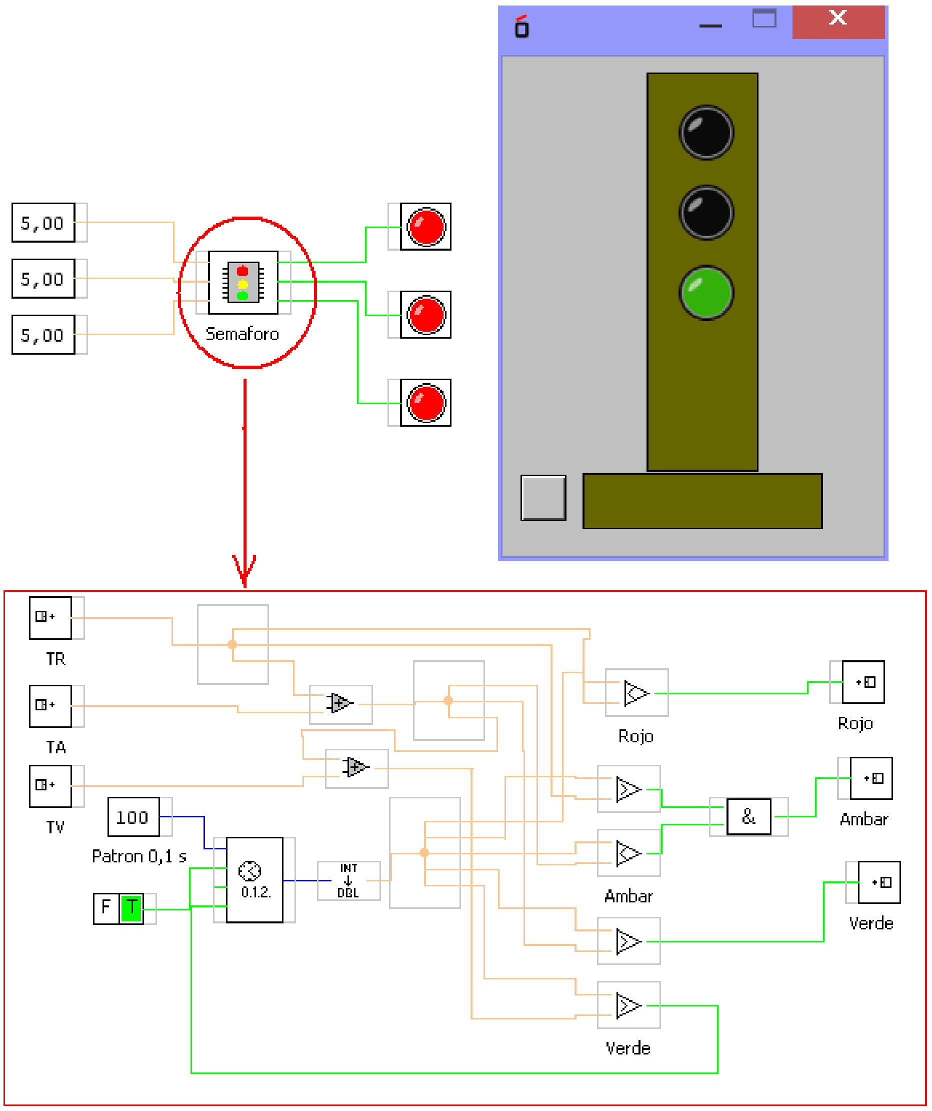


Figure 50