Introduction

The Logic Basic is a basic programming language to facilitate the development of programs in the Windows environment, to develop the programming logic in children, youth and adult beginners in computing, but is also ideal for professional programmers who want to develop programs more quickly and easily.

It is a simple language, fun and interesting, but with powerful features that can be used by the programmer to the extent that it is developing within the language.

The Logic Basic is very friendly, with a programming environment clear and very structured, ie it encourages the programmer to do organized programs, thus exercising their reasoning according to the most modern programming languages.

With the Logic Basic, you can develop commercial programs, graphic animation, games, audio-visual presentations, educational programs, music, and whatever else your mind can imagine.
How the Logic Basic works?

Logic Basic has a code environment where you write your program, this environment has a text box to the main code and more 6 text boxes for the code extensions. It also has the environment of windows, where the results of the program will be shown.

By activating the Run button (button with a green arrow to the right), the Logic Basic executes the command(s) of the first row, then the second, then the third and so on. That is, it will execute your program as if reading a text where you say, "Do this", "Do that," ..., in the order from the first line to the last.

The windows are places where you through the commands, writes words, draw pictures, and also receive information from the keyboard, mouse, hard disk, and so on. In them you can put background images, which are images that will be fixed in the background of the window, write texts, create command buttons, components such as text boxes, list boxes, check boxes, and so on, And may also draw figures that can be moved or changed by programming. There are also sounds that can be played at any time during the execution of your program.

Code extensions

There are also "Code extensions", which are text boxes where you can put some code snippets to better organize your program. To access the main code, you must press the code button (button containing the letter "C"), and to access the extensions, simply press one of the 6 red buttons on the right side of the code button.

You can imagine these extensions as a continuation of the main code, so that when you run the program, it will be as if they were concatenated (stitched) to the main code.

If you press one of the buttons of "code extensions", the Logic Basic shows in the text box below the code that corresponds to the respective extension. At the top right of the code window there is a text box labeled "Code Name" where you can be written a name to identify the code extension. This name is optional, it is not necessary to put it, but it is recommended to improve the clarity and logic of the program.
Saving and opening programs

The Logic Basic programs may be saved in any folder selected by the programmer, and it is advisable that you create a folder for each program, where will be written to the source code, image files, sound files, and so on, for the program. The extension of the Logic Basic programs are .Lbc, but you do not need to enter it because the extension is automatically inserted.

On the File menu are four options for creating, opening and recording of source code:

**New program:** When this button is pressed, the Logic Basic erases the code box and create a new program with the name New.Lbc.

**Open program:** The Logic Basic will present you with a dialog box containing the names of recorded programs. Enter the name of the desired program or double-click on top of it.

**Save:** When you press this button, the program that is in the text box will be written to disk with the current name, no questions asked.

**Save As:** The Logic Basic will present a dialog box for you to enter the name of the program to be recorded. If it has already been previously recorded, just double-click the program name in the list.
First program in Logic Basic

The main window of the Logic Basic is divided into rows and columns and the default value is 27 rows by 78 columns:

Our first program will place a sentence in a particular row and column (this is called "Coordinated") and write it in the window.

Writing in the main window

The first thing to do is to write a command in the code text box to tell the Logic Basic which row and column it should write or draw something. The name of this command is **Position**:

Position Line, Column

The arguments of a command must be separated by commas, and we can write one command per line, or more on the same line if they are separated by ; (semicolon). The next step then is to write a sentence in the window. For this we use the **Write** command:

Write "Any phrase ..."

Note that the phrase should be quoted so that the Logic Basic treat the words as just an argument, without the quotes, words could be confused, in the example above, with an expression or variable.

Running the first program

Let us put into practice what has been taught so far. In the text box type the following commands:

Position 11, 33
Write "Hello World!"

Then press the "Run" button, or press F5. The result should be the phrase **Hello World!** written in the center of the main window.

To close the window and return to the code text box, press F6.
Basic commands and basic resources

Message

The Logic Basic has a command called **Message**, identical to the **Write** command, except that it does not write directly in the main window, but in a small box message centered on the video monitor:

```
Message "Any phrase..."
```

Window

The command **Window** performs various settings in the main window of the Logic Basic, such as size, background color, background images, character font, and so on.

For example, to set the background color of the window as white, just inform the argument **Background**, and assign the desired color:

```
Window Background = White
```

To set a background image to the window, instead of color must enter the path and name of an image file, for example:

```
Window Background = "C:\Pictures\Landscape.jpg"
```

There are several other configuration options that can be entered on the same line in sequence, but they must be separated by commas, for example:

```
Window Res = 50,150, size = 30,100, Pos = 0,0
```

In the above example the command "Window" sets the window resolution with 50 rows by 150 columns, the window size to 30 rows by 100 columns, and position of the window on the desktop of Windows from line 0, column 0.

An important detail is that the size and position depend on the window resolution, which is calculated according to the current size of the window, so this must be set first.

Font

The **Font** command defines the character font of the window such as name, size, color, and so on. After defining the source, all texts written in the window will have the attributes defined by this command.

In Logic Basic new windows may exist beyond the main window (created by the command **NewWindow** that will explain later) and each one has its name, it was agreed that the name of the main window will be **Parent**. So to set a font attribute in the main window, you would type the command **Font**, then the window name with a dot, the name of the attribute and its value, for example:

```
Font Parent.Name = "Arial"
Font Parent.Color = Blue
Font Parent.Size = 20
Position 5,10; Write "Logic Basic"
```

If the window name is omitted in the **Font** command, the Logic Basic assumes the name of the main window (Parent), for example:

```
Font Name = "Arial"
```
**ClearWindow**

The **ClearWindow** command deletes all texts and drawings in the active window, except the background and components placed on it. This command has no argument:

```
ClearWindow
```

You can also use the **Cls** command, which is similar to the ClearWindow command:

```
Cls
```

### Comments

You can make comments in the program code, they will not produce any effect on the program, it is as if they did not exist in the program code.

To make a comment, you must write the sign `' (apostrophe) at the beginning of the line, or after any command line, all text written after this signal is ignored:

```
Write "Hello world!" 'Write the phrase "Hello World!" in the active window
```

In the example above, the text written after the command `Write "Hello World"` will be ignored.

### Putting multiple commands on the same line

When you put `;` (semicolon) after a command, you can then write another command on the same line. This improves the clarity of the program, and also decreases the number of lines. In fact, for the Logic Basic, each command separated by semicolon is considered as a new line. That way, you can put multiple commands on one line, for example:

```
Position 11,33; Write "Hello, World!"
```

### Commands to terminate the program

The commands **EndProgram** and **EndWindow** terminates the execution of the program in the line where they are placed.

The command **EndProgram** just ends the program, but retains the main window active. The command **EndWindow** ends the program and closes all active windows. Note that you must choose only one of them to terminate the program.
Variables declaration

In Logic Basic you can create variables, which are memories to store text, numbers and characters. There are three types of variables in Logic Basic: **String**, **Integer** and **Decimal**.

The type **String** should be used to declared variables or text string; The type **Integer** must be used to declare variables of type integer numeric to vary -2,147,483,648 to 2,147,483,647; The type **Decimal** should be used to declare variables of decimal number that can range from -1,797,693,134,862,323 to 1,797,693,134,862,323 for positive values, and the 4,940,656,458,412,470 to 4,940,656,458,412,470 for positive values. Simplifying, variables of type Decimal can be used to store both integer values as fractions, and are used primarily for working with currency values.

To declare variables you should use the keyword **Variable** or simply **Var** and then the variable name followed by its type. If not informed the variable type, the Logic Basic will assume to be of type String.

You can declare multiple variables on the same line, for example:

```logic
Variable CustomerName String, Age Integer, Salary Decimal
```

or

```logic
Var CustomerName String, Age Integer, Salary Decimal
```

After the variables have been created, you can assign text and numbers for the variables using the = (equal) operator, for example:

```logic
CustomerName = "John Robinson"
Age = 50
Salary = 1234.50
```

Note that text must be enclosed in quotation marks, we will see why below...

When you assign text to a variable, the text must be enclosed in quotation marks. In most programming languages, a sequence of letters or characters are called **Strings**. From now on, treat texts as Strings to facilitate dialogue, and also you get to be familiar with this term. Therefore, when referring to **Strings** we are referring to texts, letters, characters, and numbers in text form.

When a number is quoted, is treated as a String, and when unquoted, is treated as a numeric value, it means that we can calculate with it.

When a number is assigned to a variable explicitly, if the number is fractional, decimal places should be separated by a decimal point, never use a comma, this is a rule.

The value of a variable can be assigned to another variable, for example:

```logic
X = 7
V = X
```

In the above example, the variable X received the number 7. Then the variable V received the value of the variable X, which is 7. Therefore, the content of the variable V is equal to 7.

For the Logic Basic, a number, or a variable containing a number, are seen as numbers. That is, 7 is a number, and X is a number. Only you can not see the number that is in X, but the Logic Basic yes, because it is in his memory. The same happens with strings, when a text is in a variable, the Logic Basic will work with its content and not its name. Let us illustrate with the following program:

```logic
Write CountryName
```

When you run the program, the result is the following word: **CountryName**. The Logic Basic wrote
"CountryName" because that word is not a variable. Now let's modify the program to the following code:

```logic
Variable CountryName String
CountryName = "Canada"
Write CountryName
```

When you run the program, the result is as follows: **Canada**. The Logic Basic wrote "Canada" because the word "CountryName" was declared as a variable to the Logic Basic, and knowing this, he writes the variable content on the active window.

The name of a variable can not contain spaces in your body, so in the example above, the variable **CountryName** was not written **Country Name**. Variable names can not have the same name of the commands, keywords and reserved variables, and it is recommended that do not contain the name of commands in your body.

### Global and Local Variables

The Logic Basic has the concept of Global and Local variables. Global variables can be read at any point in the program, they retain their value within the main code, extensions, subroutines and functions. Local variables must be declared only within functions (or subroutines), and they retain their value only within the function where it is declared and after the execution of the function, they are destroyed by Basic Logic. This is important because it saves memory and avoids confusion in your program. Local variables can have the same name in different functions without causing conflicts.

#### Rule for local variables declaration

To declare a local variable must meet one rule: they can only be declared in the first lines of a function, eg:

```logic
Function Sum() Integer
Variable X Integer, Y Integer
Variable Ret Integer

X = 10; Y = 20
Ret = X + Y

Return Ret
EndFunction
```

In the above example, the variables **Ret**, **X** and **Y** are local variables. If a variable is declared in the middle of the code of a function, it will be considered as Global, for example:

```logic
Function Sum() Integer
Variable X Integer, Y Integer

X = 10; Y = 20

Variable Ret Integer

Ret = X + Y

Return Ret
EndFunction
```

In the above example the variables **X** and **Y** are local, and the variable **Ret** is global.
Operators

The function of the variables is not only store texts and numbers, they can be modified at any time by the programmer, through operators, commands and new assignments. In Logic Basic we can work with four types of operators: Arithmetic operators, increment and decrement operators, comparison and logical.

Arithmetic operators:
- + Sum
- * Multiplication
- / Division
- % Rest of division
- ^ Exponentiation

Increment and decrement operators:
- ++ Adds 1 to himself
- -- Decreases 1 of himself
- += Adds a number to it
- -= Decreases a number of it

Comparison operators:
- = Equal
- < Less than
- > Greater than
- <= Less than or equal
- >= Greater than or equal
- <> Different

Logical operators:
- And Logical conjunction
- Or Logical disjunction

Union operator:
- & Join strings or variables

The = operator is a comparison operator only when it's inside a comparison expression, in other situations it's an assignment operator.
Conditional statements and flow control

If conditional statement

This is a statement that exists in all programming languages, and its function is to make the execution of a code snippet according to the result of an expression: if the expression is true, it executes the code snippet, if it is false does not run, or run other code snippet. The name of this statement is If, and it has two more auxiliary statements: Else and EndIf. The statement Else runs another code snippet if the result of the expression is false, and the statement EndIf must always be placed at the end of a clause If, for example:

Variable X Integer

X = 1
If X = 1
    Write “X is equal to 1!”
Else
    Write "X is not equal to 1!"
EndIf

The command Else is not mandatory, but the command EndIf must necessarily be placed to finalize their respective command If.

You can place a command If within another command If, or even cascading, for example:

Variable X Integer, Y Integer

X = 1
Y = 2
If X = 1
    Write “X is equal to 1"
    If Y = 2
        Write "Y is equal to 2"
    Else
        Write "Y is not equal to 2"
    EndIf
Else
    Write "X is not equal to 1"
EndIf

GoTo directional command

As seen previously, Logic Basic executes the program line by line, from the first to the last line. But you can direct the execution of the program to another program line with the command called GoTo, which causes program execution to go to a certain line, for this you must create a label (any word followed by a colon) in a particular line of the program, and direct the execution to the line of the label with the command GoTo, for example:

Variable X Integer

X = 2
If X = 1
    GoTo LABEL1
Else
    GoTo LABEL2
EndIf
While flow control statement

The Logic Basic has a flow control statement named **While** that makes a snippet of code runs repeatedly while the result of an expression is true. Each **While** statement should have its respective **Loop** statement, which should be placed at the end of the snippet of code to be executed. When the result of the expression is false, the program execution will be directed to the next line after the **Loop** statement, for example:

```plaintext
Variable X Integer

X = 0
While X < 10
    Write X
    X++
Loop
```

The above example will write the value of X while it's less than 10, it means that he will write all the values 0-9.

You can also put a statement **While** within another statement **While**, or even cascading, for example:

```plaintext
Variable X Integer, Y Integer

X = 0
While X <= 3
    Write "X = ", X
    X++
Y = 0
While Y <= 2
    Write "Y = ", Y
    Y++
Loop
Write ""
```

LABEL1:
Write "Label 1"

LABEL2:
Write "Label 2"
Logical operators

Logical operators **And** and **Or** can be used in the command line of **If** or **While** to perform logical operations between expressions. The operator **And** performs a logical operation between expressions so that it returns a true result if all expressions are true at the same time, at least one of the expressions is false, it returns false. The operator **Or** performs a logical operation between expressions so that at least one of the expressions is true, it returns a true result, and returns false only if all expressions are false. You can perform logical operations with various expressions on the same line and also put more than one operator on the same line.

Below, examples of logical operators:

**Variable** X Integer, Y Integer

X = 1  
Y = 1

**If** X = 1 **And** Y = 1 'Returns True  
Write "True"  
Else  
Write "False"  
**EndIf**

**If** X = 1 **And** Y = 2 'Returns False  
Write "True"  
Else  
Write "False"  
**EndIf**

**If** X = 1 **Or** Y = 2 'Returns True  
Write "True"  
Else  
Write "False"  
**EndIf**

**If** X = 2 **Or** Y = 2 'Returns False  
Write "True"  
Else  
Write "False"  
**EndIf**
Arrays

Arrays are a set of variables that have a unique name, and each element of an array has an index identification. An array is declared as follows: the array name with a numerical value in parentheses and their type:

```
Variable VarName(20) String, Age(20) Integer
```

The above command line creates two arrays: CustomerName with 20 variables and Age with 20 variables. The index of an array starts with the value 0 (zero), so in the example above, the index ranges 0-19. The following is an example of how to put and get text and numbers in arrays:

```
Variable VarName(20) String, Age(20) Integer

VarName(0) = "Priscila"
Age(0) = 10
VarName(1) = "John"
Age(1) = 20
VarName(2) = "Sarah"
Age(2) = 16
```

In this example values were assigned only to arrays with index 0-2, but you can assign values to arrays with index 0-19.

You may have noticed that two arrays were created with the same amount of indexes, and each index of the array VarName was associated with the same index number of the array Age. This type of association is important because it allows you to get various information, of a person for example, with a single index.

To get a value from an array, you should write the name of the array and its index in parentheses, eg:

```
Write VarName(2)
```

If you put this command at the end of our previous example, the result will be Sarah.

In the array index you can also place numerical variables, which as its name says, being variables, it makes arrays a powerful scheduling feature. Below, an example of listing the names on the window using a variable as an index of the arrays:

```
Variable VarName(20) String, Age(20) Integer

X = 0
While X <= 2
    Write VarName(X), " - ", Age(X)
    X++
Loop
```

When running the program for this example, will be listed in the active window the first three names with their ages.

An important rule is: in the array index you can put numbers or variables, but not complex expressions. If you need to put an expression in the index of the array, assign it to a variable and then enter it as an index.
Subroutines, Functions and Timers

Subroutines and functions are features that allow the programmer to create routines (code snippets that perform a particular function), and can be executed at any time within the main program. The difference between subroutine and function is that the latter can return values, while the subroutine simply executes a piece of code without returning values.

To create a subroutine or function, you first must declare them, and this should always be done after the command `EndProgram` or `EndWindow`. If they are placed within one of the code extensions is not necessary to inform the command `EndProgram` in the main code, because the Logic Basic enters this command automatically at the end of the main code when the program runs.

Subroutines

To declare a subroutine, you must type the statement `Sub`, then the name of the subroutine, and open-close parenthesis, as shown below:

```
Sub Test()
  'Code of the subroutine
EndSub
```

Note that for every command `Sub`, you must write your respective command `EndSub`, which indicates where the subroutine ends.

After you have declared the subroutine, you should write your code, which must be between the command `Sub` and `EndSub`, for example:

```
Sub Test()
  Write "Hello martian!"
EndSub
```

This subroutine named `Test`, will write the phrase "Hello martian!" in the active window, to run it, write your name followed by open-close parenthesis at any point in the program, eg:

```
Test()
Write "End of the program!"
EndProgram
```

In the above program the subroutine was called in the first line, then the Logic Basic runs their code, writing `Hello martian!` in the active window, then returned the program execution to the next line after the subroutine and wrote phrase `End of the program`.

Functions

A function can take arguments (parameters) and also return values or strings, while subroutines do not have these resources.

To declare a function, you must write the statement `Function`, then open parenthesis and write the names of the arguments and their type (if there are arguments), then closes the parenthesis and write the type of variable it returns (if it will return values or strings), eg:

```
Function Example(Argument1 Type, Argument2 Type) Type
```

For example, let's create a function to add two integer values to be passed as arguments, and returns the sum:
Function Sum(X Integer, Y Integer) Integer
Variable Ret Integer

Ret = X + Y
Return Ret
EndFunction

Note that in the above example for each declared function you must declare their respective Command EndFunction that indicates where the function ends. Note also that we use a command called "Return", which returns a value (or string).

An important detail is that the returned value should be of the same declared type of the function, which in our example is of type Integer.

When we declare the arguments and their types, for each function call, the Logic Basic will create variables for these arguments, which will be destroyed after the end of the execution of the function, ie they are local variables, and the name of these arguments can be equal to the name of the arguments of other functions without causing conflicts.

When performing a function, you must assign your return to a variable, or pass it as an argument to another command or function of the Logic Basic, and the arguments must be informed in parenthesis, and in the same order they are declared, for example:

Variable T Integer

T = Sum(5, 2)
Write "The value of the sum of 5 + 2 is equals to ", T
EndProgram

Function Sum(X Integer, Y Integer) Integer
Variable Ret Integer

Ret = X + Y
Return Ret
EndFunction

Timers

Timers are subroutines that perform its code at certain intervals of time, without the need for you to call them in the program code. Are useful to monitor events that occur in the program in real time, for example, check what is being typed in a text box, or even create a watch that is active while other routines of the program are executed.

To declare a Timer, you should write the statement Sub, then the word Timer, sub-line and the timer name, then type the value of the interval (in hundredths of a second) in parenthesis, for example:

Sub Timer_Test(50)

There is also a command with the same name (Timer), which has two methods: Activate and Off. In method Activate you should pass as argument the value of the interval in hundredths of a second, for example:
In the above example the timer Test was disabled and then enabled again with a new value for the interval (20 hundredths of a second).
Debugging Programs

Debug is synonymous with "Take Defects", "Troubleshooting", "Remove bugs" of the computer program. The Logic Basic offers a resource to facilitate this work, and it was created the Debug button. When pressed this button, will activate the "debug mode". Thus, when the program is executed, the number of the editor, the line number, and line code, will be shown in a small window during program execution.

Thus, the programmer will visualizing step by step what the Logic Basic interpreter is running, ie you will see the line of code, and their result on the window. With this feature, it's easy to detect any errors in the program because if the Logic Basic does not run the program in accordance with what is expected, surely there is some error in the line of code where the error occurs. The programmer must then check if there is a misspelled command, check if the variables were declared correctly, and so on.
Components

Components are objects that can be created on the windows, such as text boxes, command buttons, option buttons, check boxes, calendars, scrollbars, images, etc.. There are also virtual components which can not be displayed in the windows, only the results such as the components "Rep", "Print" and "SQL."

Each type of component has a command to its creation and manipulation. The components have properties and methods. Properties are characteristics or aspects that can be assigned to a particular component or returned by him. Methods are commands that are executed by the component to perform certain tasks in relation to himself. For you to understand better, we give some examples of properties and methods:

Properties:

Visible: Sets the component as visible or invisible.
Enabled: Sets the component as active or inactive.
Txt: Sets or returns the text of a component.
BackColor: Sets the background color of a component.

Methods:

Select: Selects the text of a text box.
Add: Adds an item in the component.
SetFocus: Moves the focus to the component in question.
Remove: Removes the component of the window.

Every component has a name to characterize it, so two components, despite being of different types, can not have the same name. As a good programming technique, you can put an abbreviation of the component type as a prefix of their name, for example:

Text TxtTest, 3, 3
Button BtnTest, 7, 3

In the above example, the command Text will create a text box named TxtTest on line 3, column 3, and the command Button will create a button named BtnTest on line 7, column 7.

To assign a value to a property of a particular component just type the command name, the component name followed by a point, the property name, and then assign a value to the property with the sign = (equal), for example:

Text TxtTest.BackColor = Blue

In the above example, the color Blue is assigned to the property BackColor of the component TxtTest.

To execute a method of a component, the rule is pretty much the same, except that an action will be executed in relation to the component, for example:

Text TxtTest.Select

In the example above, the text contained within the text box component TxtTest will be fully selected.

In some components that have text or values, such as text boxes, scroll bars, radio buttons, calendars, etc.. the Logic Basic creates a variable for each component to store their values, and the name of this variable is the name of the component followed by the property name, for example, to get the text typed in the text box in our example just read the value of the property Txt as follows:

YourName = TxtTest.Txt
You can work with these properties in your program the same way you work with normal variables.

**Variables or arrays for component names**

You can create variables and vectors for component names, and this is a feature that transform the components into very versatile tools for programming.

To create variables or arrays for the name of components there is a rule: The name of the variable or array must start with the character "$", and must be of type String, for example:

```plaintext
Var $Customer String, $Card(3) String
```

That done, you can assign names to variables or arrays, for example:

```plaintext
$Customer = "Customer1"
$Card(0) = "Card0"
$Card(1) = "Card1"
$Card(2) = "Card2"
```

To create components with variables or arrays, you simply pass them in the argument Name of the components, for example:

```plaintext
Text $Customer, 5, 15
Picture $Card(0), "Card0.jpg"
Picture $Card(1), "Card1.jpg"
Picture $Card(2), "Card2.jpg"
```
Reserved variables

The Logic Basic has some reserved variables described below, so you can not create variables, functions and subroutines with names similar to these, not to cause conflicts in the program:

RetStr String, RetValue Decimal, MouseX Integer, MouseY Integer, MouseClick String, Blue Integer, Red Integer, Green Integer, Black Integer, White Integer, Yellow Integer, LightBlue Integer, DarkBlue Integer, DarkRed Integer, DarkGreen Integer, DarkYellow Integer, FieldReturn Integer, TotalFonts Integer, WindowsDir String, Desktop String, ProgramDir String, WindowsPrograms String, TestButton String, LBDir String, ObjectValue String, ReferenceValueString, SQLPointer String, LinSize Decimal, ColSize Decimal, GotFocus String, LostFocus String, DefaultBackColor Integer, RefIndex Integer, CommandLine String, StartWindowInvisible Integer, ScreenHeight Integer, ScreenWidth Integer