**LANGUAGE OVERVIEW**

LexiCom Compiler Programming Language is derived from C, VB.net and Swift Language's where the advantages involve its concise and simple syntax.

**GENERAL RULES**

1. The program must have a *Lead* function. The word *Lead* is followed by a colon “**:**”.
2. The *Lead* function must begin with the word *Start* and end with the word *End.*
3. *LexiCom* is white space insensitive.
4. *LexiCom* is a case sensitive Programming Language. Thus, *Automata*  and *AuToMaTa* are two different identifiers.
5. Declarations can be made locally or globally.
6. There is no specific arrangement in declaring.
7. *Var* is used for declaring a variable and *Let* for constants.
8. *Task* definitions are placed after the *Lead* function. To use these *Tasks* they must first be declared before the *Lead* function.
9. Reserved words must not be used as an identifier.
10. Identifiers must start with a capital letter. It must not begin with a special characters or a small letter.
11. Identifiers must have a minimum of 1 characters, and a maximum of 12 characters
12. Underscores “**\_**” are allowed in naming an Identifier as long as it is not the first character.
13. Statements are terminated with a period “**.**”.
14. The whole program must be terminated by the word *End* followed by the terminator period “**.**” then the number sign “#”.
15. Statements after the number sign “#” will not be read.

**STRUCTURE OF THE PROGRAM**

Before we introduce the basic building blocks of LexiCom programming language, let us look a bare minimum LexiCom program structure so that we can fully understand the Language in the next pages

LexiCom Hello World Example

A LexiCom program basically consists of the following parts:

* Declarations:
  + Task Declarations
  + Object Declarations
  + Global Variable Declarations
* Lead Function
* End of the program
* Task Definition

|  |
| --- |
| ***[<Task declaration>]***  ***[<Object declaration>] [<Global variable declaration>]***  ***Lead: Start  [Statement\_1.]***  ***[Statement\_2.]  …***  ***[Statement\_n.] End. [<Task definition>]#*** |

Let us look at a simple code that would print the words *“Hello World”:*

**Lead:**

**Start**

**Say “Hello World!”.**

**End.#**

Let us look at the various parts of the program:

1. The first line of the program, *Lead:* is the main function where the program execution begins.
2. The next line *Start,* it serves as the open curly brace “{” in C.
3. The next line *Say “…”* which displays *“Hello World!”* on the screen while execution of the program.
4. The last line *End.#* terminates the whole program.

**RESERVED WORDS**

|  |  |  |
| --- | --- | --- |
| **C Language** | **Proposed Language** | **Description** |
| main | Lead | Function called at program startup |
| **Data Types**  A data type is a set of [data](http://searchdatamanagement.techtarget.com/definition/data) with values having predefined characteristics. | | |
| Int | Int | A data type that contains whole numbers. |
| double | Double | Data types that contains whole numbers and a decimal part. (64-bit) |
| char | Char | A data type that deals with characters. |
| String | String | A data type that represent a series of characters. |
| Bool | Boolean | A data type can only handle two values: TRUE or FALSE. |
| Void | Null | A return type that will not return a value. |
| Const | Let | has the value and type of the constant value it represents |
| - | Array | Used to declare an array |
| - | of | Use to specify the size of an array |
| - | Task | To declare a function |
| **Data Structure**  A data structure is a particular way of organizing [data](https://en.wikipedia.org/wiki/Data_(computing)) in a computer so that it can be used[efficiently](https://en.wikipedia.org/wiki/Algorithmic_efficiency). | | |
| Struct | Object | A collection of variables placed under a single name. Variables can have different data types |
| **I/O Statements**  I/O Statements are statements used in order to | | |
| Printf | Say | Formats and prints a series of characters and values |
| Scanf | Read | Reads data and stores it in a location. |
| **Conditional Statement** | | |
| If | If | Executes a sequence of statements if the evaluation of conditional expression is true. |
| Else If | Or | Extends an IF statement to execute a different IF statement in case the condition if the IF evaluated to be false. |
| Else | Otherwise | Extends an IF statement to execute a differentstatement in case the condition if the IF evaluated to be false. |
| Switch | Option | It allows to test the equality of the value of a variable against a list of values stated in each OPTION statements. |
| Case | State | serves as the cases |
| Default | Default | Default statement is used when there is no true condition among the options |
| Break | Stop | A keyword used to terminate the execution of the iteration |
| Continue | Skip | A keyword that is used to jump to the increment and decrement of the iteration. |
| **Looping Statements** | | |
| For | For | Causes its body of statements to be executed provided ‘termination-condition’ is true. |
| While | Until | Executes the given statements in the body as long as the expression is equal to TRUE. |
| While | LoopIf | Executes the given statements in the body as long as the expression is equal to TRUE. Used right after the DO statement. |
| Do | Do | Executes the statements in the body once |
| **Supporting Statements** | | |
| Clrscr | Clear | A keyword RESET used to clear the screen |
| Return | Response | Terminates the current function and returns specific value to the caller function |
| True | Yes | Boolean True value |
| False | No | Boolean False value |
| = | is | Used to initialize/equalize a value |
| - | Start | Serves as a start of a task |
| - | End | Indicates the end of a task |
| - | Var | Used to declare a variable |
| - | Let | Used to declare a constant |
| - | by | Used in declaring a multi-dimensional array |

**RESERVED SYMBOLS**

1. ***Arithmetic Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | adds two operands |
| - | subtract second operands from first |
| \* | multiply two operand |
| / | divide numerator by denominator |
| % | remainder of division |
| ++ | Increment operator increases integer value by one |
| -- | Decrement operator decreases integer value by one |

1. ***Relational Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | Check if two operands are equal |
| != | Check if two operands are not equal. |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| >= | check left operand is greater than or equal to right operand |
| <= | Check if operand on left is smaller than or equal to right operand |

1. ***Assignment Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | Equalize a value  (Equal) |

1. ***Logical Operators***

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | Logical AND |
| || | Logical OR |
| ! | Logical NOT |

***IV. Other Symbols***

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | | | **Description** |
| . | | | Terminator (Period) | |
| , | | | String Concatenation (Comma) | |
| ; | | | Separator (Semi-colon) | |
| ‘ ‘ | | Used in defining the char value (Single quotes) | | |
| ” “ | | Used in defining the string value (Double quotes) | | |
| : | | Used in declaring a function (Colon) | | |
| ( ) | Used inarithmetic, conditions and parameter list. (Parenthesis) | | | |
| [ ] | Used in declaring an array (Brackets) | | | |
| @ | Used in accessing the element of an object | | | |