Generalidades del Lenguaje

El Lenguaje Interpretado Factory (LIF) está diseñado para ser fácil de aprender, leer y programar. La estructura de las cláusulas siguen el comportamiento esperado en la mayoría de los lenguajes de alto nivel y están en Español, intentando que se puedan leer en lenguaje natural. El LIF solo define inicialmente solo cuatro tipos de datos básicos (**Cadena**, **Numero**, **Fecha** y **Logico**), un tipo de datos de arreglo (**Vector**) y un tipo genérico (**Object**).

Una Expresión LIF consta de una serie de valores (literales), variables y funciones relacionadas por medio de operadores que son utilizadas en conjunto con Sentencias (**Definir**, **Asignar**…) y Cáusulas (**Si**, **Mientras**…) para realizar alguna rutina simple.

Declaración y Asignación de Variables:

Para declarar una variable se usa la palabra clave **Definir**, seguida del tipo de datos y finalmente la lista de nombres de variables a definir (separados por punto y coma). Para asignar un valor a una variable se usa la palabra clave **Asignar**, seguida del nombre de la variable, el operador de asignación (**=**) y el nuevo valor. Si el tipo de datos asignado no concuerda con el tipo de datos de la variable, se intentará una conversión implícita, si la conversión falla se disparará un error.

```
Definir TipoDato NombreVariable1 [; NombreVariable2...]
Asignar NombreVariable1 = Valor1
Borrar NombreVariable1
```

No se permite declarar dos veces la misma variable; Cuando ya no sea necesaria una variable es recomendable borrar su definición usando la sentencia **Borrar**, con lo cual se libera la variable y su valor en memoria; esto último es especialmente recomendable cuando la variable es de tipo **Objeto** o **Vector**.

Tipos de Datos:

Tipo	Ejemplo de uso
Cadena	`Definir Cadena lcVariable` `Asignar lcVariable = "prueba" & " cadena"`

Numero	`Definir Numero lnVariable`
	`Asignar lnVariable = 15`
	`Asignar lnVariable = Truncar(lnVariable/4)`

Fecha	`Definir Fecha lfVariable`
	`Asignar lfVariable = #20090101#`
	`Si ObtenerMes(lfVariable) > 6`
	`  //Ejecutar código`
	`FinSi`

Lógico	`Definir Logico llVariable`
	`Asignar llVariable = EsVacio('')`
	`Si llVariable OR ObtenerMes(Hoy())>6`
	`  //Ejecutar código`
	`FinSi`

<table>
<tr><td>Vector</td><td>

```
Definir Vector lvVariable
Definir Cadena lcPrimerColor
Asignar lvVariable = Vector.mCrear('azul','verde','rojo')
Si Vector.mLongitud(lvVariable) >= 1
      //Obtiene le primer color (azul)
      Asignar lcPrimerColor = lvVariable[1]
SiNo
     Asignar lcPrimerColor = 'desconocido'
FinSi
```

</td></tr>
<tr><td>Objeto</td><td>

```
Definir Objeto loVariable
Definir Cadena loVariable.lcNombre
Definir Fecha loVariable.lfFecha
Asignar loVariable.lcNombre = 'Objeto Nuevo'
Asignar loVariable.lfFecha = Hoy()

Si (loVariable.lnNuevaPropiedad > #20090101#)
      //Ya termino el año 2008
FinSi
```

</td></tr>
<tr><td colspan="2">Observaciones:

- El tipo de datos **`Vector`** almacena una colección de valores de tipo **`Objeto`**, donde cada item puede ser accedido por medio de un índice (con base 1) entre corchetes.
- Para el tipo de datos **`Objeto`**, está planeado implementarlo con un comportamiento similar al de `Object` en JavaScript: el objeto es dinámico, lo que significa que puede agregarse propiedades nuevas por medio de la sentencia **`Definir`**.
- Los primeros tipos de datos (simples) tienen cada uno una representación literal (por ejemplo, las cadenas entre comillas). Se implementará igualmente una representación literal para los tipos de datos **`Vector`** y **`Objeto`**; preliminarmente será algo como esto:

```
//Para evitar usar una funcion como Vector.mCrear()
Definir Vector lvVariable
Asignar lvVariable = ['azul','verde','rojo']
//Para evitar declarar y asignar cada propiedad individualmente
Definir Objeto loVariable
Asignar loVariable = {lcNombre='Objeto Nuevo', lfFecha=Hoy()}
```

</td></tr>
</table>

Tabla 1. Tipos de datos

Estructuras de Control

ESTRUCTURA CONDICIONAL SI

Se inicia con la palabra clave **Si** seguida de una expresión lógica. Si la expresión lógica devuelve **TRUE** se ejecutan las instrucciones internas del **Si**, sino continúa la ejecución en la línea siguiente a la palabra clave **FinSi**.

```
Si ExpresiónLógica1
      //Instrucción 1
SinoSi ExpresiónLógica2
      //Instrucción 2
SinoSi ExpresiónLógicaN
      //Instrucción N
Sino
      //Última Instrucción
FinSi
```

Tabla 2. Estructura Condicional Si

La estructura condicional **Si** admite más de una ejecución condicional, usando la palabra clave **SinoSi** para indicar una nueva expresión lógica a ser evaluada en caso de que las expresiones de las cláusulas anteriores hayan resultado en **FALSE**. Adicionalmente se admite una clausula **Sino**, por medio de la palabra clave Sino, que es ejecutada incondicionalmente cuando todas las condiciones anteriores hayan resultado en **FALSE**.

ESTRUCTURA CONDICIONAL COMPARAR.

Se inicia con la palabra clave **Comparar** seguida de una expresión. El valor de la expresión es luego comparado con las expresiones de comparación de las cláusulas **Es**; si alguna de estas comparaciones devuelve **TRUE** se ejecutan las instrucciones asociadas. Si ninguna de la expresiones devuelve **TRUE** se ejecutan las instrucciones asociadas a la cláusula opcional **Es Otro**.

```
Comparar ExpresiónAComparar
      Es Comparacion1
            //Instrucción 1
      Es Comparacion2
            //Instrucción 2
      Es Otro
            //Última Instrucción
FinComparar
```

Tabla 3. Estructura Condicional Comparar

La estructura condicional **Comparar** debe contener al menos una clausula **Es**, y opcionalmente terminar con una clausula **Es Otro**. Una vez que se ha ejecutado el bloque de instrucciones de la cláusua **Es** apropiada, o En caso de que ningúna de las cláusulas **Es** se ejecute, la ejecución continúa en la instrucción inmediata posterior a la cláusula **FinComparar**. Las expresiones de comparación pueden ser de diferentes tipos, según el ejemplo siguiente:

```
Comparar ObtenerDia(Hoy())
      Es 1 //un valor
            //Instrucciones para 1
      Es 2; 3; 5 //una lista de valores
            //Instrucciones para 2, 3 o 5
      Es Desde 7 Hasta 10 //Un rango de valores
            //Instrucciones para 7 a 10 (ambons inclusive)
      Es < 13 //Una comparación directa
            //Instrucciones para menores a 17
      Es 14; 16; Desde 18 Hasta 23; > 26 //Una lista combinada
            //Instrucciones para 14, 16, de 18 a 23 y mayores a 26
      Es Otro //cualquier otro caso
            //Instrucciones para el resto de los casos
FinComparar
```

Tabla 4. Ejemplo de Comparar

ESTRUCTURA ITERATIVA PARA

Se inicia con la palabra clave **Para** seguida de la variable que se va a incrementar en la iteración. Las cláusulas **Desde** y **Hasta** definen respectivamente el valor inicial y el valor final que va a tomar la variable. La cláusula opcional **Paso** define de cuanto en cuanto será aumentada o disminuida la variable de iteración (de forma predeterminada se incrementa en 1 unidad en cada itreración). Una vez completadas las iteraciones del bucle **Para** se ejecuta la instrucción inmediatamente posterior a la cláusula **FinPara**.

Opcionalmente se puede incluir la cláusula **Continuar**, que fuerza a iniciar la siguiente iteración antes de llegar a la cláusula **FinPara**. También se puede incluir la cláusula opcional **Salir** para detener la ejecución del bucle antes de que la variable numérica lacanze el valor final.

```
Para VariableNumerica Desde Valor1 Hasta Valor2 [Paso Valor3]
      //Instrucción 1
      //Instrucción 2
      [Continuar]
      //Instrucción 3
      [Salir]
      //Instrucción N
FinPara
```

Tabla 5. Estructura Iterativa Para

La variable numérica debe haber sido definida previamente como de tipo **Numero**. Los valores **Desde**, **Hasta** y **Paso** de la estructura **Para** deben ser de tipo **Numero** (aunque no necesariamente enteros), de no serlo se intentará una conversión implicita, disparando un error en caso de que no sea posible realizar la conversión.

```
Definir Numero lnContador
Definir Numero lnSumador
Para lnContador Desde 2 Hasta 20 Paso 2
      //Obtiene la suma de los enteros de 2 en dos del 2 al 20.
      Asignar lnSumador = lnSumador + lnContador
FinPara
```

Tabla 6. Ejemplo de Estructura Iterativa Para

ESTRUCTURA ITERATIVA PARACADA

Se inicia con la palabra clave **ParaCada** seguida la variable que se va a iterar, seguida de la cláusula **En** y la colección sobre la cual se hace la iteración. Una vez completadas las iteraciones del bucle **ParaCada** se ejecuta la instrucción inmediatamente posterior a la cláusula **FinParaCada**.

Opcionalmente se puede incluir la cláusula **Continuar**, que fuerza a iniciar la siguiente iteración antes de llegar a la cláusula **FinPara**. También se puede incluir la cláusula opcional **Salir** para detener la ejecución del bucle antes de que el bucle halla recorrido todos los elementos de la colección.

```
ParaCada Variable En Colección
      //Instrucción 1
      //Instrucción 2
      [Continuar]
      //Instrucción 3
      [Salir]
      //Instrucción N
FinParaCada
```

Tabla 7. Estructura Iterativa ParaCada

La variable a iterar debe haber sido definida previamente del mismo tipo de datos que los elementos de la colección, de no serlo se intentará una conversión implicita, disparando un error en caso de que no sea posible realizar la conversión.

```
Definir Numero lnContadorCorto
Definir Numero lnContadorLargo
Definir Vector lvTabla
Definir Objeto loFila

Asignar lvTabla = Tabla.mSeleccionar("SELECT * FROM articulos")
ParaCada loFila En lvTabla
      //Obtiene el total de artículos con código muy corto.
      Si Cadena.mLongitud(loFila.cod_art) < 5
            Asignar lnContadorCorto = lnContadorCorto + 1
      SiNo
            Asignar lnContadorLargo = lnContadorLargo + 1
      FinSi
FinParaCada
```

Tabla 8. Ejemplo de Estructura Iterativa ParaCada

ESTRUCTURA ITERATIVA MIENTRAS

Se inicia con la palabra clave **Mientras** seguida de la expresión lógica que se va a evaluar antes de iniciar cada iteración. El bloque de instrucciones que será repetido en cada iteración está delimitado por la cláusula **FinMientras**.

Opcionalmente se puede incluir la cláusula **Continuar**, que fuerza a iniciar la siguiente iteración antes de llegar a la cláusula **FinPara**. También se puede incluir la cláusula opcional **Salir** para detener la ejecución del bucle antes de que la variable numérica alcanze el valor final.

```
Mientras ExpresiónLógica
      //Instrucción 1
      //Instrucción 2
      [Continuar]
      //Instrucción 3
      [Salir]
      //Instrucción N
FinMientras
```

Tabla 9. Estructura Iterativa Mientras

El siguiente ejemplo muestra como usar un bucle **Mientras** para calcular el factorial de un número.

```
Definir Numero lnContador
Definir Numero lnFactorial
Asignar lnContador = 7
Asignar lnFactorial = 1
```

```
Mientras lnContador > 1
      //Obtiene el factorial de un número dado.
      lnFactorial = lnFactorial * lnContador
      lnContador =  lnContador - 1
FinMientras
```

Tabla 10. Ejemplo de Estructura Iterativa Mientras