Sentencias en LabVIEW*

Patxi Alkorta Egiguren

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

Este módulo proporciona la información necesaria para poder utilizar las sentencias de selección, secuenciación y repetición de LabVIEW.

1 Sentencias de selección

Se trata de un tipo de sentencia que permite ejecutar un código dado u otro, según el valor que tenga la variable de entrada de la sentencia de selección. Este tipo de sentencia se llama **Case Structure** y se encuentra en **Functions/Programming/Structures**. Esta sentencia puede ser utilizada para dos casos, o para más de dos o casos múltiples.

En el ejemplo que se muestra en Figure 1, se pretenden obtener solo los valores aleatorios entre 0 y 0'7 a partir de la función que genera números aleatorios entre 0 y 1 (**Functions/Programming/Numeric/Random Number**). Entonces, se recurre a la función de comparación **Greater or Equal?** de la paleta **Functions/Programming/Comparison**, ya que ésta proporciona una salida booleana **true**, si 0'7 es mayor o igual que el valor aleatorio generado, y **false**, si no es así. En este sentido, la salida en el caso true será el valor aleatorio generado, y en caso contrario, 0'7 , ya que en ese caso el valor aleatorio es superior a éste.

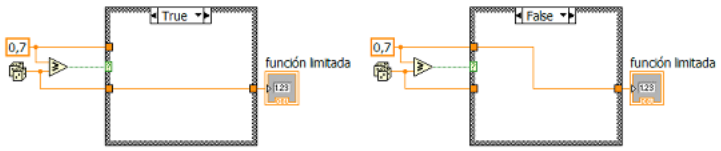

Figure 1

Para los casos de más de dos entradas, es decir, los de entradas múltiples, se inserta una sentencia case en el diagrama. Inicialmente éste está configurada para el primer caso, con lo que tendremos que modificarlo. En primer lugar, se cambia la entrada de tipo booleano a tipo constante entero: se crea una constante de tipo entero en el diagrama desde **Functions/Programming/Numeric/Numeric Constant**. Si se desea que este elemento pase a ser un control, solo hay que sustituirlo por un elemento de ese tipo. Seguidamente,

*Version 1.1: Oct 24, 2008 11:46 am -0500
†http://creativecommons.org/licenses/by/2.0/

clickando sobre el **Selector Label** con el botón derecho se elige la opción **Duplicate Case**, donde al repetir la operación se van añadiendo los caso 2, 3, 4 . . ., Figure 2. Ahora solo queda incluir en cada caso el módulo de código correspondiente.

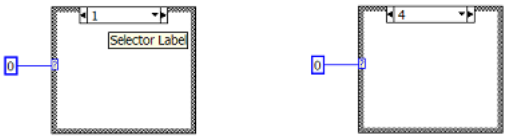

Figure 2

Para borrar uno de los casos, se repite la operación, pero ejecutando en ese caso **Delete Case**. La Figure 3 muestra el panel resultado final en funcionamiento.

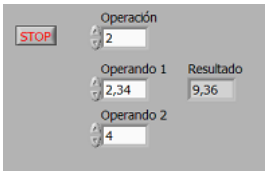

Figure 3

2 Sentencias de secuenciación

Se trata de un tipo de sentencia que garantiza la ejecución secuencial (uno detrás de otro) de varios módulos de código. Cuando en el diagrama de **LabVIEW** son definidos varios módulos de código, en realidad no se sabe en qué orden van a ser ejecutados, es decir, no se sabe cuál de ellos se ejecutará en primer lugar y cuál en el último lugar. Hay aplicaciones en las que este orden de ejecución no es importante, pero en otras es de vital importancia. Por ello, en las aplicaciones donde el orden de ejecución de los distintos módulos es crítico, resulta imprescindible utilizar las sentencias de secuenciación para garantizar esa secuencia de ejecución de los módulos.

La herramienta de desarrollo de aplicaciones **LabVIEW** ofrece dos sentencias de secuenciación. La primera es la **Flat Sequence Structure** y se encuentra en **Functions/Programming/Structures**. Cuando es insertada en el diagrama de nuestra aplicación, aparece con una única ventana o **Frame**: es la unidad o elemento donde irá un módulo dado del código de nuestra aplicación. Para añadir más ventanas, se pulsa sobre la ventana con el botón derecho del ratón y se ejecuta **Add Frame Before** o **Add Frame After**, para añadir la ventana antes o después de la actual, respectivamente. Veamoslo con un ejemplo. Pensemos en dos operaciones aritméticas, suma y producto, que han de realizarse con dos operandos independientes en cada caso, donde primero se tiene que realizar la suma y luego el producto, Figure 4. Notar que las conexiones de los operandos han de realizarse desde los controles o exterior, hacia el interior de las ventanas.

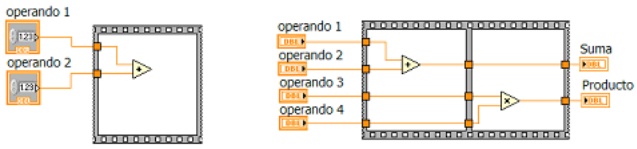

Figure 4

Cuando son necesarias muchas ventanas, porque así lo requiere la aplicación, aparece el problema del espacio en el diagrama, de tener que desplazar varias pantallas desde el inicio de la secuencia hasta su final. Entonces, lo lógico es sustituir la sentencia **Flat Sequence Structure** por la sentencia **Stacked Sequence Structure**, la cual funciona exactamente igual que la anterior, solo que apila las ventanas una encima de otra. Aunque a simple vista no sea posible ver la secuencia, resulta muy práctico su uso cuando el espacio disponible es pequeño. Repitamos el ejercicio anterior con esta nueva sentencia.

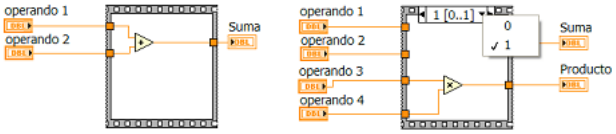

Figure 5

En este caso, al añadir la segunda ventana después de la primera, automáticamente aparece un índice para seleccionar cada una de las ventanas y acceder a su código, Figure 5.

3 Sentencias de repetición

Se trata de sentencias que permiten la ejecución repetida de un módulo de código dado, mientras se cumpla la condición de ejecución de la sentencia (condición de permanencia en la sentencia). Los hay de dos tipo: **for** y **while**.

3.1 For

La sentencia iterativa for, al igual que el resto de las sentencias, se encuentra en **Functions/Programming/Structures**. Se trata de una sentencia que repite N veces la ejecución del módulo de código que se encuentra en su interior. Este parámetro, es fijado bien mediante una constante o bien mediante un control, y está asocido al índice i de la sentencia. El parámetro i parte de 0, y tras ejecutarse el código de la sentencia y comprobarse que aún es menor que N, se incrementa en una unidad y se repite la operación. Así hasta que i es igual que N, entonces, deja de ejecutarse la sentencia y el código que ella contiene. Veámoslo con un ejemplo sencillo donde N=3 y el código consiste en multiplicar por 1 el valor del índice **i** y visualizarlo, Figure 6.

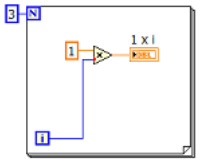

Figure 6

Al poner en ejecución esta aplicación, se inicializan los dos parámetros de la sentencia, N=3 y i=0, y a continuación comienza a ejecutarse la parte repetitiva o iterativa:

1. como se cumple que i<N, 0<3, entonces se ejecuta y se visualiza 1xi=1x0=0. A continuación se incrementa i, es decir, i=i+1=0+1=1
2. como se cumple que i<N, 1<3, entonces se ejecuta y se visualiza 1xi=1x1=1. A continuación se incrementa i, es decir, i=i+1=1+1=2
3. como se cumple que i<N, 2<3, entonces se ejecuta y se visualiza 1xi=1x2=2. A continuación se incrementa i, es decir, i=i+1=2+1=3
4. como no se cumple que i<N, 3<3, entonces no se ejecuta el código, ni tampoco se incrementa i, con lo que deja de ejecutarse la sentencia for.

Así, el resultado final de la ejecución de este ejemplo queda como se puede ver en la Figure 7. Entonces, un ciclo for se ejecuta N veces y su indice i se va incrementando desde 0 a N-1.

Figure 7

Es posible anidar varias sentencias **for**, es decir, introducir sentencias **for** con sus módulos de código en otras sentencias **for**. Quizás, el caso más usual de sentencias anidadas sea el de una sentencia **for** dentro de otra **for**, o si no, dentro de una sentencia **while**. P.e., si se desea que la aplicación anterior se repita 4 veces, entonces lo que se había hecho para el ejemplo aterior es introducido dentro de una nueva sentencia cuyo parámetro N lo fijamos a 4, tal y como se vé en la Figure 8.

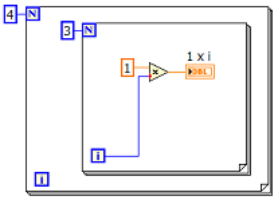

Figure 8

3.2 While

La sentencia de repetición **while** ya se ha descrito brevemente en la primera parte de este curso. Se trata de una sentencia que permite ejecutar de forma repetida el código que se encuentra definido en su interior mientras se cumpla la condición de permanencia o de ejecución de esta sentencia. La condición de permanencia es un elemento de tipo bit, es decir, de tipo booleano, y para que la sentencia **while** ejecute el código que se encuentra en su interior, el valor del elemento booleano de permanencia ha de ser de **false** (falso). En el momento en que este elemento adquiera el valor **true** (verdadero), la condición de permanencia desaparece y el ciclo **while** deja de ejecutarse. Se dice que se sale del bucle **while**, con lo que deja de ejecutarse el código que se encuantra en su interior.

Normalmente, la condición de permanencia suele ser el resultado de una operación lógica, o bien, un pulsador de tipo **STOP**. Otro elemento que contiene la sentencia **while** es el índice **i**. Este elemento indica el numero de veces que se ha ejecutado el código que se encuentra en el interior de la sentencia.

Un elemento que se suele utilizar mucho a la hora de comprobar el funcionamiento en tiempo de ejecución es el temporizador **Wait (ms)**. Este objeto se encuentra en **Functions/Programming/Timing** y lo que hace es insertar un retardo temporal definido en milisegundos (ms) fijado en su entrada (lado izquierdo). El ejemplo que se muestra en Figure 9, muestra la versión del ejercicio anterior, Figure 8, pero con una sentencia externa de tipo **while**, donde se ha insertado un retardo un retardo de 1 segundo en la ejecución del ciclo **for** interno.

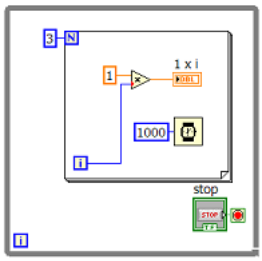

Figure 9

4 Nodos de realimentación y Registros de desplazamiento

Los nodos de realimentación y registros de desplazamiento son objetos que están relacionados con las sentencias de repetición. Se trata de elementos que se utilizan en módulos de código donde algún o algunos valores pasados son utilizados en el cálculo actual. P.e., si queremos obtener el valor actual de un parámetro donde para obtenerlo hace falta sumar un 2 al valor anterior del mismo, es decir **y=y+2** (**léase y_actual= y_anterior+2**), entonces es necesario utilizar un nodo de realimentación o si no un registro de desplazamiento, tal y como se ve en la Figure 10 derecha.

Para introducir un nodo de realimentación en el diagrama Figure 10 izquierda, se inserta un objeto **Feedback Node** desde **Functions/Programming/Structure**, cuya salida se conecta con la otra entrada del sumador, y la salida de éste con la entrada del nuevo objeto. A acontinuación, clickando con el botón derecho del ratón sobre el nuevo objeto, se ejecuta **Move Initializer One Loop Out**, con lo que el elemento que permite insertar el valor inicial del valor anterior, se desplaza a la sentencia repetitiva, donde se le asigna un valor inicial, en este caso 3.

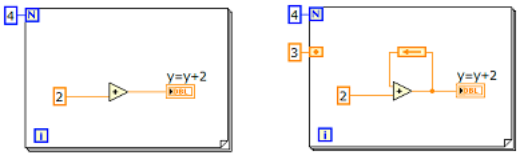

Figure 10

Para el caso del registro de desplazamiento, también se parte del diagrama de Figure 10 izquierda del caso anterior, solo que ahora nos ubicamos con el ratón en la parte derecha de la sentencia **for**, justo en el límite, y clickando con el botón derecho ejecutamos **Add Shift Register**. Seguidamente aparecen dos flechas, Figure 11, donde la de la izquierda permite asignar el valor inicial del valor anterior, igual que en caso anterior, y es conectada a la otra entrada del sumador. En cuanto a la flecha de la derecha, la salida del sumador se conecta a ella.

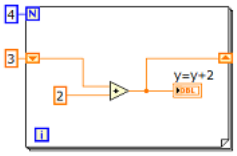

Figure 11

Cuando la ecuación a implementar tiene varios valores anteriores, p.e., el valor anterior y su anterior, **y(k-1)** e **y(k-2)**, respectivamente, siendo el valor actual **y(k)**, entonces hay que añadir más elementos a la izquierda de la sentencia **for**, clicando sobre la flecha que se encuentra ahí y ejecutando **Add Element**. Cada nuevo elemento insertado es anterior al anterior y se puede inicializar de forma individual. P.e., el siguiente diagrama, Figure 12, representa a la ecuación en diferencias **y(k)=y(k-1)+2+3*y(k-2)**.

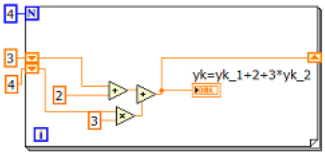

Figure 12

5 Ejercicios propuestos

Exercise 1

Desarrollar un instrumento virtual que proporcione el valor de una de las cuatro operaciones aritméticas con dos operandos, según el valor del control **Operación** (si es 0, suma, si es 1, resta, si es 2, producto, y si es 3, división). La aplicación dispondrá de un pulsador STOP para que deje de ejecutarse.

Exercise 2

Realiza un instrumento virtual que genere las primeras 10 coordenadas y, u ordenadas, de una recta del plano Cartesiano, cuya ecuación es **y=m x + d**, siendo **m** la pendiente positiva, **d** el origen, positivo o negativo, y **x** la ordenada abcisa positiva. Utiliza la sentencia for.

Exercise 3

Construye un VI que genere X números aleatorios entre 0 y 7 con la función **Random Number**. X está condicionado al primer número generado fuera del rango pedido.

Exercise 4

Construye un VI que genere X números aleatorios entre 4 y 8 con la función **Random Number**. X está condicionado al primer número generado fuera del rango pedido.

Exercise 5

Implementar la ecuación en diferencias **y(k)=[4 − 3*y(k-1)+ 53*y(k-2)-34*y(t-3)]/7**, siendo los valores iniciales los siguientes: **y(k-1)in= -2; y(k-2)in= 6.79; y(k-3)in= 12,09**;