

## 17. Parámetros

Durante el tema anterior pudiste comprobar la potencia que ofrecen las variables en Benu: Comprobaste que pueden almacenar datos con diversas características, incluso grandes cantidades de ellos, el principal límite de todo ello es nuestra imaginación.

El tema que viene a continuación pretende darle todavía más utilidad al concepto de variable en Benu, puesto que aprenderemos a ofrecer a los distintos procesos los valores que queremos que tomen para su ejecución. En primera instancia podremos comunicar a un proceso aspectos básicos como cuál es su gráfico o su posición a la hora de invocarlo, pero también veremos que podemos ofrecerle información más compleja como las estadísticas de fuerza, agilidad o inteligencia que queremos para él, de la misma forma que se haría en un RPG.

### 17.1 Paso de parámetros

Como ya sabes, la primera línea de la **declaración** de cualquier proceso en Benu tiene la siguiente estructura:

```
PROCESS nombre_del_proceso (  
...  
BEGIN  
...  
END
```

El paréntesis de apertura y cierre que añadimos junto al nombre del proceso tiene una utilidad especial, ya que en él podemos enumerar una serie de **parámetros** que el proceso **recibirá** a la hora de ser invocado.

Por ejemplo, en los parámetros podremos indicar que el proceso va a recibir un entero indicando su posición inicial o una string indicando su nombre. ¿Cuál es la utilidad de todo esto?, pues que gracias a ello, con una única **declaración** de proceso podremos lograr distintos comportamientos según cuáles sean los parámetros que reciba a la hora de ser invocado.

Para ilustrar todo esto con un ejemplo, vamos a diseñar un sencillo proceso disparo que recibirá como parámetros los valores de gráfico inicial y final sobre los cuáles deberá animarse.

Para ello indicaremos en la declaración del proceso que recibiremos 2 valores enteros (De tipo int) a los cuales llamaremos graph\_ini y graph\_fin. (Debes notar que el nombre de los parámetros lo damos nosotros, verás que son algo muy similar a las variables.)

La declaración de nuestro proceso disparo quedaría así:

```
PROCESS disparo ( int graph_ini , int graph_fin )
```

...

Fíjate que los parámetros se añaden con la misma sintaxis con la que se declaraban variables, esto es una gran ventaja, ya que nos permite declarar procesos que reciben parámetros de cualquier tipo.

Ahora la **invocación** del proceso cambiará, ya que cada vez que lo invoquemos deberemos darle los valores enteros que necesita para graph\_ini y graph\_fin. Ejemplos de invocación serían los siguientes:

```
disparo ( 1, 10 );
```

```
disparo ( 25, 40 );
```

...

## 17.2 Dando utilidad a los parámetros

Antes de darle el uso adecuado a esos nuevos valores graph\_ini y graph\_fin, comprueba que el programa compila correctamente.

Comprobarás que la ejecución no muestra ninguna diferencia de comportamiento, y es que hemos añadido parámetros, pero no hemos incluido dentro del proceso disparo las instrucciones adecuadas que nos permitan sacar provecho de ellos.

Lo que queríamos conseguir con los nuevos parámetros era algo similar a:

“Cuando reciba los valores 1 y 10 (Por ejemplo), el disparo se animará dentro de ese intervalo de gráficos, mientras que cuando reciba los valores 25 y 40 se animará dentro de ese otro intervalo.”

Dentro del bloque BEGIN - END del proceso disparo podemos referirnos a graph\_ini y graph\_fin de la misma manera que nos referimos a cualquier variable, de hecho cualquier parámetro de un proceso se comporta como una variable PRIVATE del mismo, para cada proceso invocado puede tener distintos valores.

```
PROCESS disparo ( int graph_ini , int graph_fin )
BEGIN
    graph = graph_ini;          //El proceso toma como valor de gráfico el dado por el parámetro graph_ini
    ...
    LOOP
        graph = graph + 1;      //El proceso suma 1 al valor de graph para animarse
        IF ( graph > graph_fin ) //Si graph se hace mayor que el valor de graph_fin
            graph = graph_ini;  //Entonces vuelve a valer lo que vale graph_ini
        END                      //Fin de la condición
    ...
    FRAME;
END
END
```

Para que comprendas mejor el funcionamiento del código anterior, piensa que invocamos el disparo de la siguiente forma:

```
disparo ( 1 , 10 );
```

El resultado de la ejecución del disparo anterior sería equivalente al de este proceso:

```
PROCESS disparo ( )
BEGIN
    graph = 1;                  //El proceso toma como valor de gráfico el dado por el parámetro graph_ini
    ...
    LOOP
        graph = graph + 1;      //El proceso suma 1 al valor de graph para animarse
        IF ( graph > 10 )       //Si graph se hace mayor que el valor de graph_fin
            graph = 1;          //Entonces vuelve a valer lo que vale graph_ini
        END                      //Fin de la condición
    ...
    FRAME;
END
END
```

Observa que simplemente hemos sustituido graph\_ini y graph\_fin por 1 por 10. De la misma manera, el resultado de la invocación del proceso disparo pasándole como parámetros los valores 25 y 40 sería el equivalente a sustituir graph\_ini y graph\_fin por 25 y por 40.

Si has entendido esto enhorabuena, ya puedes invocar cualquier disparo sencillo haciendo uso del mismo proceso simplemente cambiando los valores que se le pasan como parámetros.